PS: Polygon Streams

A Distributed Architecture for Incremental Computation Applied to Graphics

Rajiv Gupta

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-03

PS: Polygon Streams
A Distributed Architecture for Incremental Computation Applied to Graphics

Rajiv Gupta

April 24 1987

Caltech-CS-TR-88-03

Abstract

Polygon Streams is a distributed system with multiple processors and strictly local communication. A unique custom VLSI chip that constitutes an independent processing module forms a stage of the PS pipeline. The number of these modules in PS is a variable that is determined by the application. PS features a modular architecture, multi-ported on-chip memory, bit-serial arithmetic, and a pipeline whose computation can be dynamically configured. The PS design closely subscribes to the system characteristics favored by VLSI.

The task of scan conversion is very intensive in computation and pixel information access for rendering computer graphics images on raster scan displays. It is very coherent and suitable, however, for forward difference algorithms. The discrete and regular layout of the raster display in conjunction with the largely local effect of a pixel on an image, make rendering amenable to parallel architectures with localized memory and communication. These are precisely the attributes favored by VLSI and typical of PS.

A modification of the Digital Differential Analyzer is implemented to Gouraud Shade and depth buffer convex polygons at high speeds. The scan conversion task is distributed over the processors to efficiently subdivide the image space and maximize concurrency of processor operation.

A study of the tradeoffs and architectural choices of the PS reveal the merits and deficits of the PS approach in comparison with Pixel-Planes, Super-Buffers, and SLAMs.

Contents

List of Figures

Chapter 1

Introduction

Computer Graphics has traditionally subscribed to two primary goals: realism and speed. The first goal attempts to conceive, create, and display scenes that are true to life. The latter goal is devoted to practicality. We wish to retain as much of the realism as possible as we make the modeling and rendering of images faster. Interactive, realistic computer graphics is our ultimate goal.

In the near future, as the scenes that we want to create get more complex and the phenomena that we want to simulate get more involved, we will continue to lack the ability to manipulate realistic images in real-time. To justify spending the time and computational resources in obtaining an exact image, we should first convince ourselves that the interactions in the scene – of objects, viewing parameters, light sources, to name a few – are as we had envisaged them. In order to make this decision with reasonable accuracy we need a way to quickly preview approximations of these scenes interactively. Real-time simulations and animation also require images containing thousands of polygons to be rendered within a frame time. The work described in this thesis renders shaded and depth-cued polygons on raster scan displays at speeds that match the requirements of these applications.

1.1 Organization of the thesis

In Chapter 2 we present the context of the PS System application in a graphics pipeline. By means of pascal pseudo-code we describe the incremental algorithm implemented by the system for performing Depth-Buffering and Gouraud Shading.

In Chapter 3 we discuss the system architecture of PS. This includes the VLSI implementation of the incremental algorithm discussed above and a detailed design of the scan line interpolation subsystem (SLI) of PS.

In Chapter 4 we elaborate on the architecture choices of PS. We study the influence

of VLSI on architecture design and in the context of this understanding we follow the architecture evolution of an SLI. We present the VLSI implementation issues on which we concentrated and explain some of the salient features of the PS design. Finally, we compare the PS architecture with that of the Pixel-Planes, Super-Buffers, and SLAM systems.

In Chapter 5 we analyze the quantitative aspects of PS. We derive the best and worst polygon and pixel rates and trace any discrepancy in the communication and processing bandwidths.

In Chapter 6 we explore the realm of application possibilities of the PS architecture. We discuss enhancements of the current architecture and include, in detail, a modification prompted by the quantitative analysis of Chapter 5.

Appendix A is a proof of the incremental algorithm.

Appendix B is a report on the fabrication details and current status of the PS chip.

Chapter 2

The Graphics Application

2.1 The Depth-Buffer Rendering Pipeline

The graphics pipeline for converting an image from the world or modeling coordinates to the screen or pixel coordinates is depicted in Figure 2.1. A polygonal object is typically

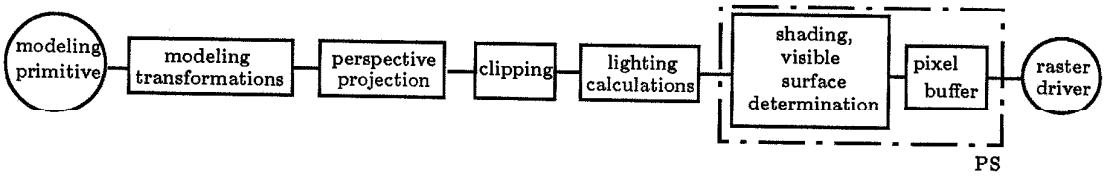


Figure 2.1: The Depth-Buffer Rendering Pipeline

described in terms of the position, x, y, and, z, and color, r, g, and b, attributes of its vertices. The viewing, clipping, perspective, and lighting computations give us a new set of vertex and color attributes in screen space. The PS system accepts the screen coordinate and color information of the vertices of a convex polygon and interpolates them to produce (z, r, g, b)-tuples for all pixels that lie within the polygon. This information is suitable for raster-scanned displays. The shading and visible surface determination task is the most computation and communication intensive component of the graphics pipeline. We render convex polygons at very high speeds by implementing two of Computer Graphics' well-known algorithms: depth-buffering [Foley 84] and Gouraud shading [Gouraud 71].

2.2 The Depth-Buffer

The depth-buffer or z-buffer, an image space algorithm, is a means of eliminating hidden surfaces. Thus at any point (x, y) in the viewing plane, we can only see the object for which the z value (or depth) at that point makes it closer to the eye than all the other objects in the screen. The implementation calls for a z-buffer that stores the z values for the pixels of the screen just as the refresh buffer stores the intensity or color values. The depth-buffer algorithm is:

During scan conversion, for all (x, y) within the polygon

1. *Calculate the smallest polygon depth at* $(x, y), z(x, y)$
   {*Note that multiple* (x, y, z)*-tuples of the three-dimensional polygon map onto every* (x, y) *"within" an edge-on polygon. The polygon point with the smallest depth at* (x, y) *is visible at* (x, y). }

2. *If* $z(x, y) <$ *z-buffer value at* $(x, y), z_{buffer}(x, y)$, *then*
   { *the polygon point is closer to the viewer than the point whose intensity is currently in the refresh buffer at* (x, y) }

   (a) *Replace* $z_{buffer}(x, y)$ *with* $z(x, y)$

   (b) *Replace pixel-buffer value at* $(x, y), RGB_{buffer}(x, y)$, *with* $RGB(x, y)$.

2.3 Gouraud Shading

Gouraud shading linearly interpolates the intensity (color) values at the vertices of a polygon to determine the intensity (color) values for points that lie within the polygon.

Consider the convex polygon of Figure 2.2, the intensity value at l, which lies on the polygon edge BA, is a linear interpolation of the intensity values at vertices B and A. Similarly, the intensity at r, which lies on the edge DE, is a linear interpolation of the intensity values at vertices D and E. l and r have the same y-value and for that value of y they constrain the polygon in the x-axis from the left and right respectively. *i.e.* for all (x, y) that lie within the polygon and $y = y_l = y_r, x_l \leq x \leq x_r$. Thus the intensity at all such (x, y) is a linear interpolation of the intensities at the edge points l and r.

2.4 The Interpolation Algorithm

In the xy plane of the 2D image, pixel values (x, y) are limited by the size of the screen, or by an integral multiple for sub-pixelling as we shall see later. Since we work in the image

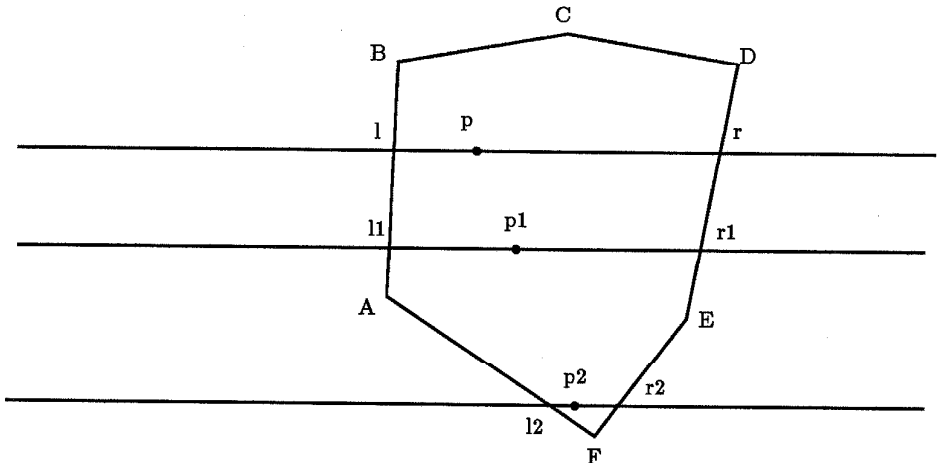


Figure 2.2: Gouraud Shading a convex polygon

space, we use integer arithmetic exclusively. For all calculations mentioned above – along a polygon edge, along a scan line, and in both these cases the interpolation of z, R, G, and B – the same algorithm is used. The interpolation algorithm is incremental and uses only integer arithmetic. It is a variation of the DDA, the Digital Differential Analyzer [Newman 79], and is best described by the pseudo-pascal function:

{for the vertices $(x1, y1)$ *and* $(x2, y2)$ *of a polygon, we want to find the x- intercepts of the polygon edge for consecutive values of y between* $min(y1, y2)$ *and* $max(y1, y2)$*. Referring to Figure 2.2 we are finding the x-intercept for all l that lie on edge BA }*

Interpolation of x against y:

if $(y1 > y2)$ *then*
    begin
        $y_{start} := y2; x_{start} := x2; \Delta y := y1 - y2; \Delta x := x1 - x2;$
    end
else
    begin
        $y_{start} := y1; x_{start} := x1; \Delta y := y2 - y1; \Delta x := x2 - x1;$
    end;

if $(\Delta x = 0)$ *then*

```
    for i := y_start to (y_start + Δy) do
        output(x_start, i)
else if (Δy = 0) then
    for i := x_start to (x_start + Δx) do
        output(i, y_start)
else
    begin

        quotient := ΔxDIVΔy;
        remainder := ΔxMODΔy;
        p_sum := 0; { p_sum/Δy is the cumulative error}
        x_current := x_start;
        y_current := y_start;
        count := Δy;
        output(x_current, y_current);

        while (count > 0) do
            begin
                x_current := x_current + quotient;
                p_sum := p_sum + remainder;
                if (p_sum ≥ Δy) then
                    begin
                        p_sum := p_sum − Δy;
                        x_current := x_current + 1
                    end;
                count := count −1;
                y_current := y_current + 1;
                output(x_current, y_current)
            end

    end;
```

Chapter 3

The Architecture and Organization of the PS System

3.1 System Architecture

The system performs two distinct but dependent interpolations: along polygon edges and along scan lines. Along polygon edges we interpolate x, r, g, b, and z against y to obtain pixel values at the edge pixels that constrain the polygon on consecutive scan lines (*e.g.* lr-pairs in Figure 2.2). Along a scan line we interpolate r, g, b, and z against x to obtain pixel values at consecutive pixels that lie within the edge pixels of that scan line (*e.g.* p on scan line lr in Figure 2.2).

For a given scan line, only after the values at the edge pixels have been calculated may the values for pixels intermediate to those edge pixels be calculated. In Figure 2.2 it is clear that we may attempt to calculate the values for p only after we have calculated the values for l and r. Note that scan line interpolation is dependent on the polygon edge interpolation only for the values of the edge pixels. Also, interpolations along different scan lines are mutually independent for the same reason. Thus in Figure 2.2, scan line interpolation to obtain the value at p given the values at l and r is independent of the polygon edge interpolation to obtain the values at l2 and r2. Both are independent of the scan line interpolation to obtain the value at p1 given the values at l1 and r1. This lends the system to a functional hierarchy – polygon edge interpolation and scan line interpolation. The PS system architecture is shown in Figure 3.1.

Since we are not constrained by any polygon processing order we can arbitrarily extend the architecture at this level. If the polygon edge interpolation (PEI) is the bottleneck we can balance the system with multiple PEIs. Conversely, if the scan line interpolation (SLI) is the bottleneck we can balance the system with multiple SLIs. A hybrid approach, as

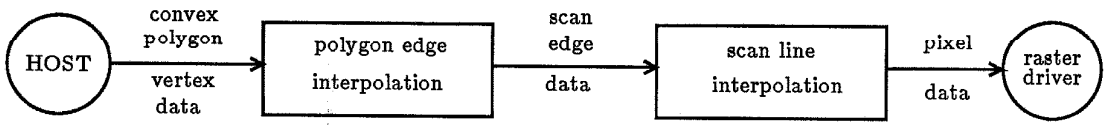


Figure 3.1: The PS System Architecture

shown in Figure 3.2 allows us to optimally balance the number of SLI's with respect to the PEI's, and the numbers of PEI's and SLI's with respect to cost-speed-accuracy indices. It is now trivial to fine tune the architecture to the scene composition. For example, scenes with consistently small polygons, as in the case wherein the accuracy in rendering complex curved surfaces is enhanced by fracturing the surface into larger numbers of smaller polygonal artifacts, require more PEIs.

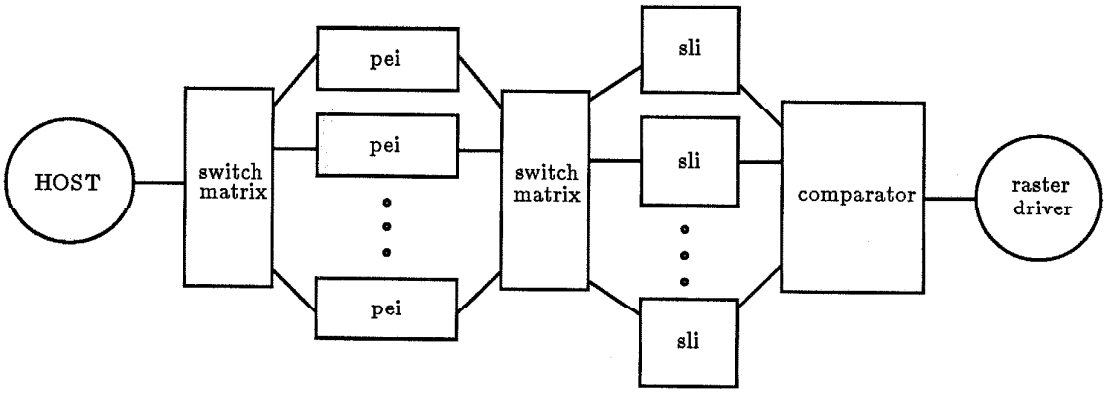


Figure 3.2: A Hybrid System Architecture tailored to the application

The potential for parallel hardware is readily apparent. This project deals with the detailed design and implementation of the scan line interpolator subsystem.

3.2 The Scan Line Interpolation Subsystem

3.2.1 Parallelism

Since the interpolations along different scan lines are independent operations, the scan line interpolator subsystem consists of several scan line interpolators which work in parallel. The best that we can do is to have an independent interpolator for each scan line. The time to interpolate the entire convex polygon is then the time to interpolate its longest scan line extent.

3.2.2 Pipelining

Interpolations along the same scan line for different polygons are mutually independent. Also, the interpolation for a scan line can be broken into discrete and repetitive steps that successively interpolate for the pixels that lie on the scan line. Thus each scan line interpolator can be pipelined so that interpolation for polygon 'a' can be started before that for polygon 'b' is completed.

For a given scan line y, we may interpolate at $(x1, y)$ for polygon 'a' concurrently with the interpolation at $(x2, y)$, $x1 \neq x2$, for polygon 'b'. Note that $(x1, y)$ and $(x2, y)$ cannot belong to the same convex polygon because the DDA insists that the interpolation at $max(x1, x2)$ is dependent on the cumulative error calculated at the interpolation of $min(x1, x2)$ if $x1, x2$ belong to the same polygon. Thus the scan line processor can be split into many sub-scan line processors each responsible for a set of m unique pixels and potentially interpolating for a different convex polygon. In Figure 3.3 both convex polygons are interpolated in parallel by a non-intersecting set of sub-scan line processors.

The sub-scan line processor stores its pixel-set information, $(z, r, g, b)_i$ for $1 \leq i \leq m$, in its private memory. This memory is organized as a pixel and depth buffer for the sub-image. Thus over the array of processors we have a distributed frame and depth buffer for the entire image.

The DDA implementation for this processor array and the data format we chose gave us an elegant, efficient, and versatile interconnection structure.

3.3 Data Structure and Algorithm Implementation

Each iteration through the inner loop of the DDA gives us the interpolated value for one pixel. It requires:

- the divisor,
- the quotient,

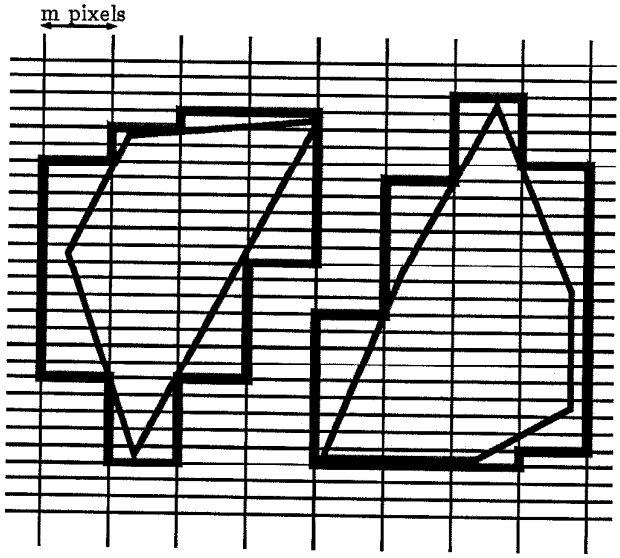


Figure 3.3: The Parallelism and Pipelining in the system: both polygons are interpolated in parallel by the pipelined scan line processors.

- the remainder,
- the last interpolated value, and
- the cumulative error.

Besides the cumulative error and the interpolated value, all the other values need to be calculated once for each scan line. These are calculated by interpolating along the polygon edges and are input from the polygon edge interpolator subsystem for each scan line. Thus the inner loop of the DDA constitutes scan line interpolation.

By pipelining the scan line interpolator into multiple sub-scan line interpolators we are effectively opening the inner loop of the DDA into multiple loops each with a default number of iterations equal to the extent of the sub-scan line processor, m in our case. Thus for a given scan line, y, sub-loop1 interpolates for $0 \leq x \leq m-1, y$; sub-loop2 for $m \leq x \leq 2m-1, y$ and so on. When sub-loop1 finishes, it puts all the required data into the registers for sub-loop2. When sub-loop2 starts working on these values sub-loop1 is now available to interpolate for the sub-scan line $0 \leq x \leq m-1, y$ of another convex polygon. Note that although the sub-loops are working concurrently, they are interpolating for different polygons. *For a given polygon and scan-line there is a definite order of sub-loop operations and a unique data packet in the pipeline.* Parts of this data packet, such as

the last interpolated value and the cumulative error get modified as the data packet moves through the sub-loop pipeline. This imposes a simple communication structure: sub-loop$n$ communicates with sub-loop$(n+1)$ to whom it sends the modified data packet, and with sub-loop$(n-1)$ from whom it receives data packets.

If we break the loop as mentioned above, in addition to the values above the data packet needs to carry x_{start} and x_{end}, the start and end values for the loop. ($x_{\text{end}} - x_{\text{start}}$ = the divisor). Note that all the sub-loops perform the same function. For sub-loop$q$ (which interpolates for $((q-1)m \leq x \leq qm-1, y)$ this would be:

if new data in input registers and $x_{\text{start}} \leq qm-1$ then interpolate for the scan line section that lies in its domain *i.e.* from $max((q-1)m, x_{\text{start}}), y$ until $min(qm-1, x_{\text{end}}), y$; if we have not reached the last pixel of the polygon on that scan line, fill the registers of sub-loop$(q+1)$ when it is ready to accept the modified data packet.

In spite of their common functionality, the sub-loops need to know their index, q, to make the comparison, $(q \quad 1)m \leq x \leq qm-1$, and are not identical. This implies that we need $ceil(X/m)$ different sub-loops (processors) for a screen size X pixels wide. This translates to extra logic and initialization overheads to simulate different sub-loops from identical implementations of the processor.

With a slight modification in the data format and functionality this limitation can be overcome. In the format above, the values of x_{start} and x_{end} were fixed with respect to the image axis and unchanged for all of the sub-loops. But the subloops do not need the absolute values of x_{start} and x_{end}; they need these values only with respect to their domain. For example subloop$q$ only needs to know that it should start interpolating from the third pixel in its domain. The connectivity information tells us that this "third" pixel of the subloop corresponds to $x = (q-1)m+3$. Thus if x_{start} and x_{end} could somehow be "normalized" with respect to the sub-loop indices, then the index information would be redundant and the sub-loops could be identical.

Conceptually, we normalize by associating a y-axis with each data packet. As it moves through the pipeline, the data packet shifts its y-axis. Thus the y-axis for a data packet shifts to the right by the sub-loop count, m, as the packet moves between successive sub-loops. As Figure 3.4 shows, by this data manipulation we are effectively moving the sub-loops to the left and keeping the modified data packet stationary. Now all the sub-loops believe that they are the starting sub-loop and hence are identical.

We implement the normalization by :

$$\begin{aligned} x_{\text{start}}(\text{to be sent}) &:= x_{\text{start}}(\text{received}) - m(\text{the loop count}) \\ x_{\text{end}}(\text{to be sent}) &:= x_{\text{end}}(\text{received}) - m(\text{the loop count}) \end{aligned}$$

$dx = x_{\text{end}} - x_{\text{start}}$, remains unchanged.

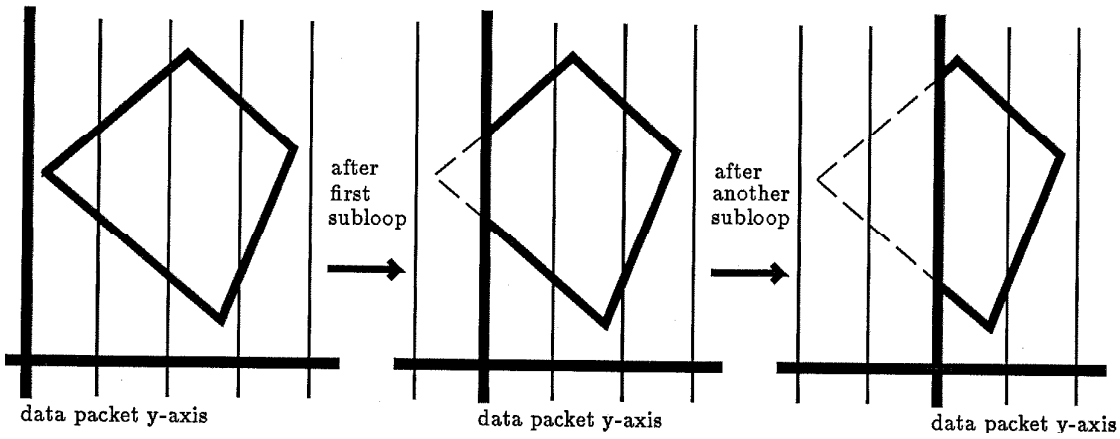
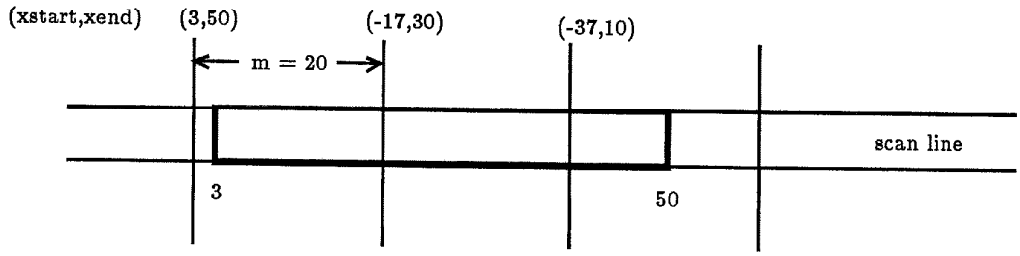


Figure 3.4: The effect of "Normalizing" a data packet



dx = xend - xstart

= 47

Figure 3.5: A numerical example to illustrate "Normalization"

Irrespective of index, all sub-loops now perform the function:

if new data in input registers and $x_{start} \leq m-1$ then interpolate for the scan line section that lies in its domain *i.e.* from $max(0, x_{start}), y$ until $min(m-1, x_{end}), y$; if the last pixel of the polygon on that scan line has been interpolated then do not send any data packet down the scan line. Otherwise, renormalize x_{start} and x_{end} for the next sub-loop; wait until the next sub-loop is ready to accept data; fill the registers of the next sub-loop.

With this normalization, the sub-loops are all identical and the index information is hardwired in the communication structure. Thus we can replicate this one generic sub-loop for displays of arbitrary size.

3.4 Translation into VLSI

3.4.1 The Two-Dimensional Chip Array

In translating into VLSI, we mapped n of these sub-scan line processors each with its private memory onto one chip. Thus each chip is responsible for a unique m by n pixel subspace. The mapping is illustrated in Figure 3.6. The tradeoffs in the choice of m and n are described in Chapter 4.

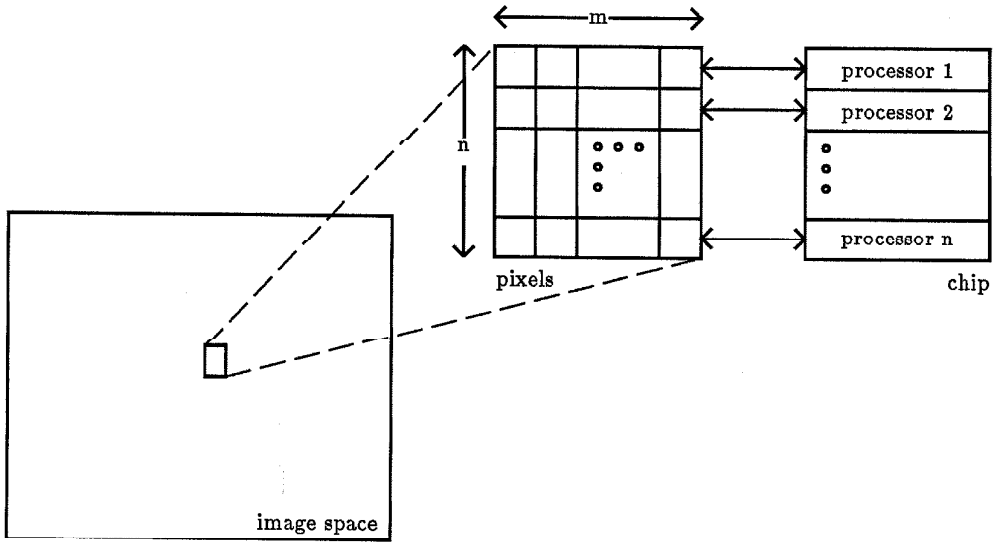


Figure 3.6: Processor-Pixel Mapping

All processors $i, 0 \leq i \leq n-1$, correspond to the same sub-loop index but for different loops (scan lines). Processor $i, 1 \leq i \leq n$, in chip(j,k) interpolates for pixels that lie in its

domain, $((j-1)m \leq x \leq jm-1, (k-1)n+i)$. Processor i in chip$(j+1,k)$ interpolates for $(jm \leq x \leq (j+1)m-1, (k-1)n+i)$ and is the adjacent subloop to processor i in chip(j,k). Thus data packets (unique to a scan line of a polygon) move from left to right in the row of chips and only the data packets for scan lines $(k-1)n \leq y \leq kn-1$ need to be input to row k starting at chip$(1,k)$.

Subpixelling is a way of reducing the aliasing effects inherent in the depth buffer algorithm. Each physical pixel is now fractured into multiple (usually a square grid) virtual sub-pixels. We interpolate in this virtual image space that is larger than the physical image space (screen space) and use an averaging filter to merge the sub-pixels when we paint a physical pixel.

For a display size of Q by R and a subpixelling size of q by r, the array of chips would have indices $1 \leq x \leq ceil(Qq/m) = X, 1 \leq y \leq ceil(Rr/n) = Y$. Each chip communicates with the two chips on either side of it in the row, *i.e.* chip$(a,b), 1 < a < X; 1 \leq b \leq Y$, communicates with chips chip$(a-1,b)$ and chip$(a+1,b)$. Chip$(1,b), 1 \leq b \leq Y$, communicates with chip$(2,b)$ on the right and with the polygon edge interpolator on the left and chip$(X,b), 1 \leq b \leq Y$, communicates only with chip$(X-1,b)$ on the left.

If the incoming data packet is for a busy processor (with all its input buffers full) then the chip passes on a busy signal to the chip on its left so that it may refrain from overwriting this data. This communication ensures that no information is lost when we have a choked pipeline. If the data is for a sub-scan line that intersects the chip subspace, then the data packet is written into the scratch registers of the corresponding processor and the chip is free to accept more data. If the first pixel, x_{start}, lies beyond the chip domain, the x_{start} and x_{end} values are normalized and the data packet is passed onto the next chip (on the right) when it becomes free to accept this data. A normalized data packet is also passed onto the next chip when an on-chip processor completes interpolating and the last pixel on the scan line, x_{end}, does not lie in its domain. If the last pixel has been interpolated then we have finished interpolating that scan line and the data packet is not passed on. Since the polygons are clipped to the image space, it follows that chip(X,y) , $1 \leq y \leq Y$, need only communicate with chip$(X-1,y)$ on the left.

Communications to the left are to indicate the busy status of the pipeline and those to the right are for transmitting scan data; this data is always normalized. The interprocessor communication is by asynchronous, source-initiated handshaking as data transfer is not lock-step with every clock tick. The communication paths for a chip array are depicted in Figure 3.7.

We have a wave of data packets moving from left to right through this chip array for each polygon. The wavefront for a polygon initiates interpolation for that polygon in all the incident processors. Polygon waves may be rhythmically pumped into the system in a systolic manner.

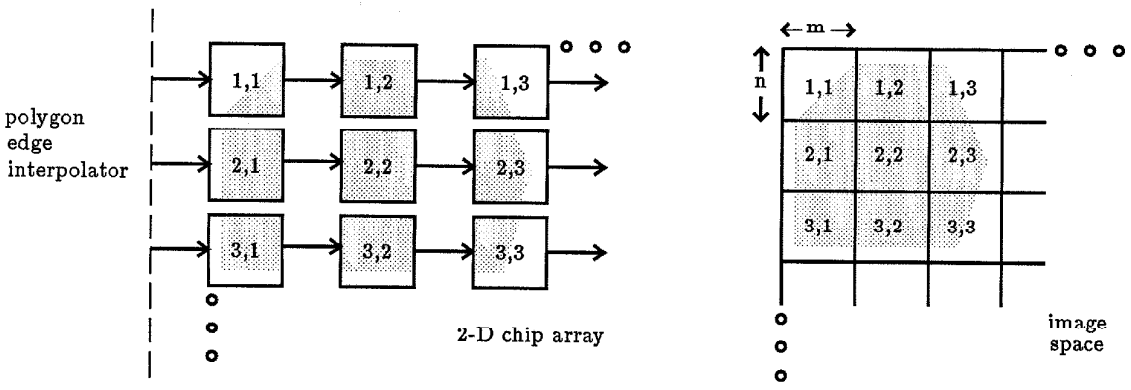


Figure 3.7: The communication structure of the chip array

3.4.2 Chip Architecture

Each chip has to handle handshaking and data communication for its n processors. The information for communicating with the chip to the left is distinct and independent of that with the right. In the former case we accept data packets and specify our status. In the latter, we send data packets and query the status of the next chip. Still another distinct source of data communication is with the external video circuitry for displaying the pixel values for the m by n pixels of this chip. This requires that we recognize a video-scanning signal and in response output the values of the current pixel as we cycle through all the m by n pixels.

The four distinct functional components are:

- n processors each with its private memory for m unique pixels.
  Each processor interpolates for the m pixels in its domain and the memory is part of the distributed pixel and depth buffers.

- Preprocessor
  The preprocessor inputs data packets and outputs its status to the chip at the left or to the polygon edge interpolator in the case of the first chip of the row. If the data packet is not for any of the chip's processors (the starting pixel lies beyond the chip domain) it passes the packet to the postprocessor.

- Postprocessor
  The postprocessor takes data packets from the n processors and the preprocessor,

normalizes them, and outputs them to the next chip. It is responsible for all the handshaking signals for this communication with the preprocessor of the next chip.

- Video-processor
  The video-processor cycles through the on-chip buffer and outputs the current pixel information in response to a video-out signal. Its access of the memory banks is transparent to, and independent of, the processors.

Additionally, each chip has a clock and timing signal generator that converts the single phase clock input to an internal two-phase non-overlapping clock. The clock generator is also responsible for producing the control signals for the preprocessor and the postprocessor and the control signals that are common to all the processor and memory banks.

This functional modularity drives the chip architecture shown in Figure 3.8.

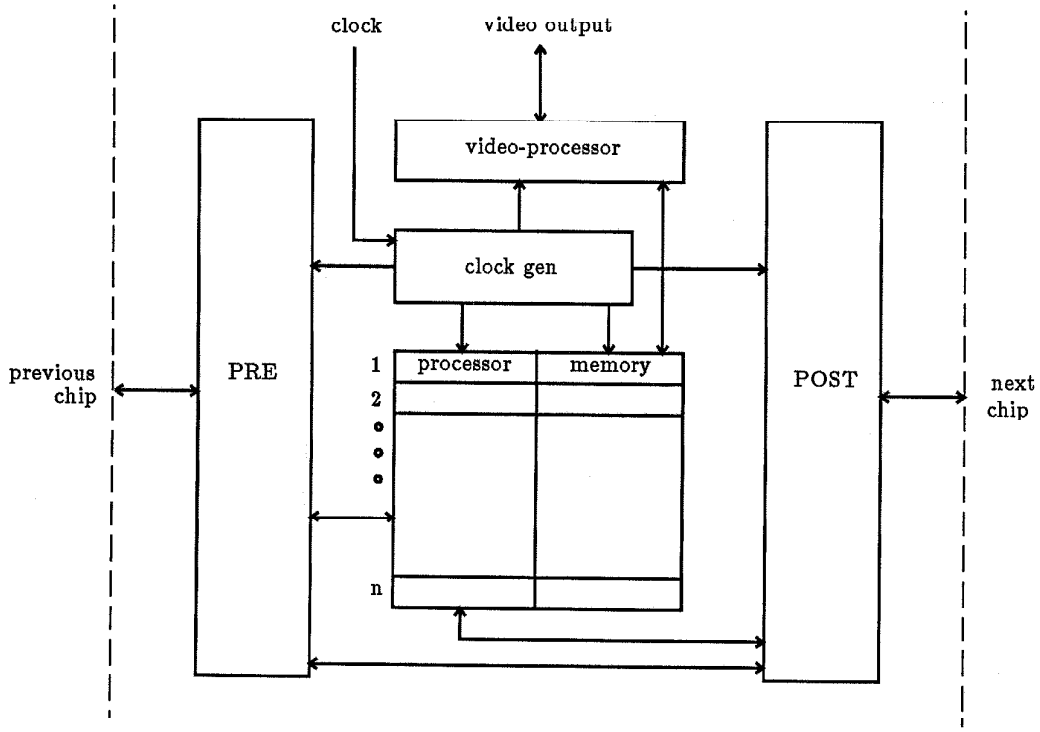


Figure 3.8: Chip Architecture

The Processor Design

All processing on the chip is bit-serial. The "processor clock period" is the time the processor takes to interpolate the values for one pixel. This requires the interpolation of r, g, b, and z; since the processor takes one system clock period to process a bit and we have $r+g+b+z$ bits, the processor clock period is $r+g+b+z$ times the system clock period.

The processor uses its scratch registers, identical and contiguous to pixel memory, exclusively for interpolation. Note that the the pixel format and the scratch register format are different; the pixel format is r, g, b, z bits while the scratch registers have to store and use the entire data packet for the interpolation. The processor access of pixel memory is only to read the existing pixel value and to either write back the same value (as the memory design below shows we have a destructive memory read-out), or write the new calculated value. We write back into the pixel memory the values that correspond to $min(z_{\text{old}}, z_{\text{new}})$, or the new values irrespective of the z-comparison result if hidden surface elimination is disabled by the command field in the data packet. By superseding the z-comparison we can initialize the buffer with an image-sized polygon having background characteristics – typically a constant color and a z at infinity.

The processor has the following functional units:

- Control unit
  This interfaces with the pre and the post processor. When the preprocessor indicates that the current data packet is for the processor, the unit fills the scratch registers with the data packet from the preprocessor, initiates the interpolation, communicates its busy status to the preprocessor, and issues all the hand-shaking signals with the postprocessor when it wants to transmit a data packet to the next chip.

- Interpolator
  This is the guts of the processor arithmetic function. The new values are calculated for r, g, b, z serially using the data packet values which are in the scratch registers of the processor. It is responsible for all communication with the pixel memory.

- z-comparator
  This determines if the pixel values in the memory are to be rewritten or to be replaced with the new values. It decodes the command bit that inhibits the z-comparison.

- x_{last} determination unit
  This controls when the processor should stop interpolating. It is a serial decrementor that starts with the number of pixels to be interpolated. This number is determined by the preprocessor.

- Formatting unit
  The Formatting unit reads values from the scratch-pad registers of the processor and

extends data or in other ways formats this data so that it can be directly absorbed by the interpolator. This unit draws its existence from our attempt to minimize the length of the data packet resulting in different formats for the same variable in pixel and scratch memories. For example, the dividend (r, g, b, z) and the divisor (Δx) values may have a different number of bits (as was in our implementation). While we store dividend bits for these quantities in the pixel memory, the remainder need only be *min*(dividend,divisor) bits and the cumulative error (to which this remainder is added) has to be divisor bits.

Figure 3.9 depicts the processor architecture.

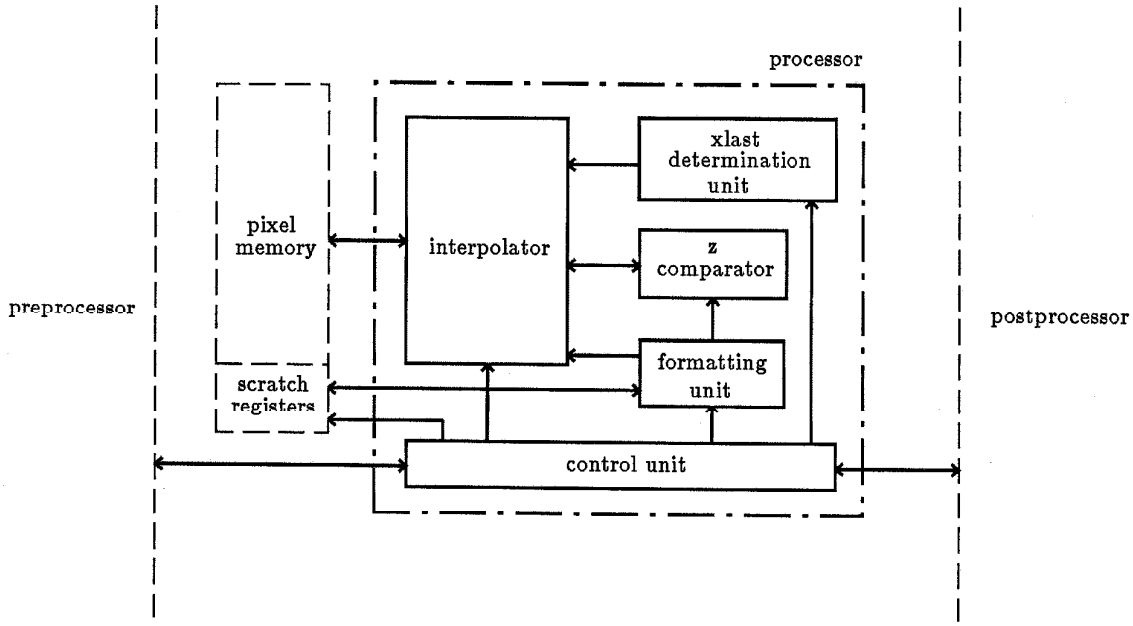


Figure 3.9: Processor Architecture

The Preprocessor Design

The preprocessor consists of the following functional units:

- Input control unit (icu)
  This handles all the hand-shaking signals with the postprocessor of the previous chip; for this it communicates with the processors and the postprocessor of the chip.

- On-chip test unit (otu)
  This determines if the chip domain intersects the scan extent. ($x_{\text{start}} \leq m - 1$).

- x_{len} calculator (xc)
  Using the x_{start} and x_{end} values this unit determines the number of pixels that should be interpolated by the current chip. x_{len} is the intersection of the scan line with the chip's domain. Thus $x_{\text{len}} = min(m, x_{\text{end}} - x_{\text{start}}, x_{\text{end}})$. When x_{end} lies within the domain of the chip (*i.e.* $x_{\text{end}} \leq m - 1$), the x_{len} calculator passes this information to the processor so that a spurious data packet is not passed on to the next chip.

- x_{start} control unit (xcu)
  This unit controls the processor access of the pseudo-random access memory. If $x_{\text{start}} \leq 0$ then the first pixel to be accessed is the first pixel in the memory of the processor; if not then x_{start} is the first pixel that the processor should interpolate.

- Processor select unit (psu)
  This unit decodes the y_{scan} value in the input data packet and activates the corresponding processor to read the data into its scratch registers.

The preprocessor architecture of Figure 3.10 manifests this functionality.

The PostProcessor Design

The postprocessor talks with the preprocessor when the input data packet is not for the current chip and has to be passed on to the next chip, to the processors when they attempt to send modified data packets to their counterparts in the next chip, and to the preprocessor of the next chip to determine its busy status. It also has to normalize the data packet by subtracting m from x_{start} and x_{end}.

Thus the postprocessor functional components are:

- Normalizing unit (nu)
  This takes care of the normalizing subtraction.

- Outbuffer control unit (obu)
  This control unit passes control of the output drivers to the preprocessor or a processor that has a data packet for the next chip. It polls the preprocessor and the postprocessor round-robin starting with the processor next in schedule to the processor that outputted last.

- Output control Unit (ocu)
  This unit controls hand-shaking with the preprocessor of the next chip.

The postprocessor architecture is shown in Figure 3.11

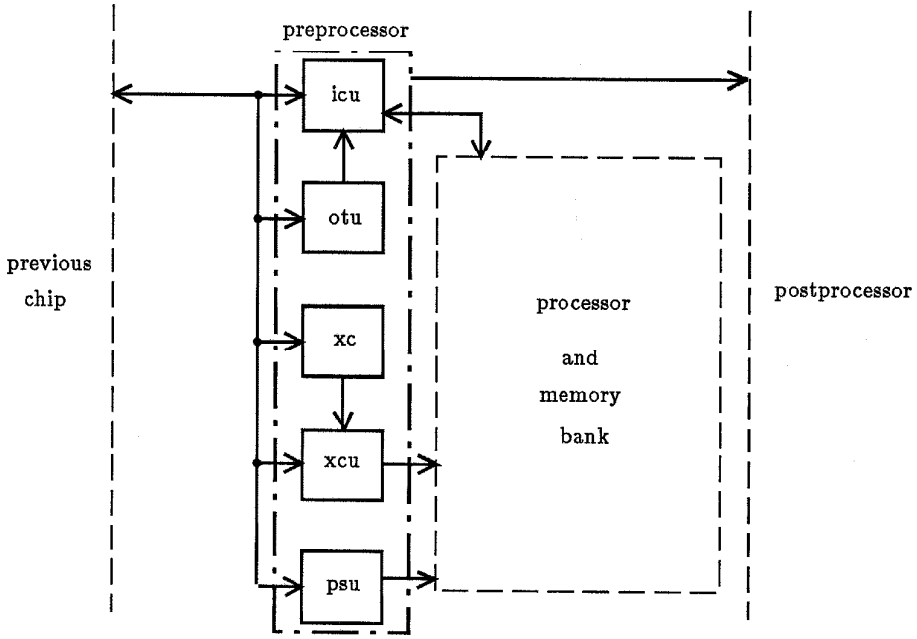


Figure 3.10: Preprocessor Architecture

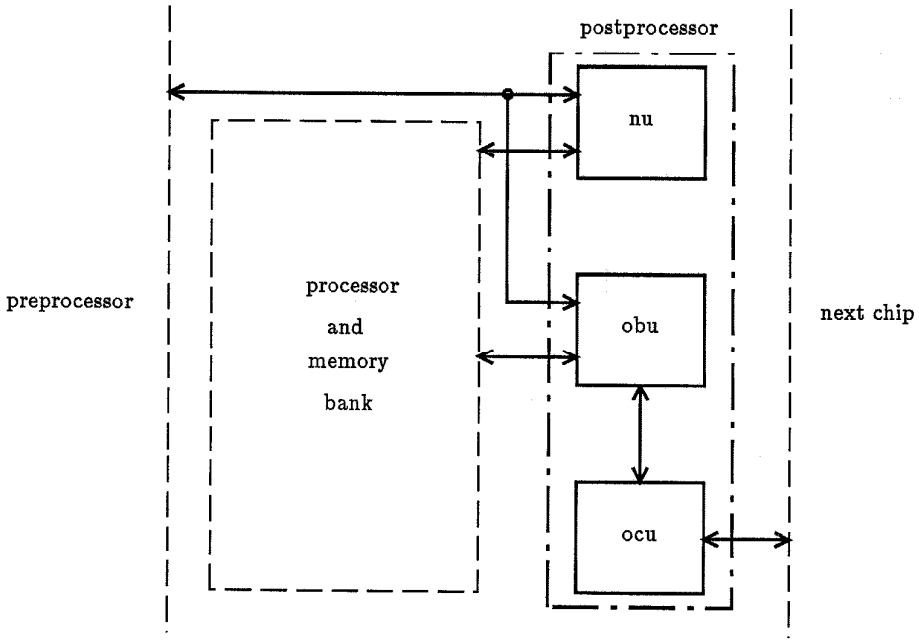


Figure 3.11: Postprocessor Architecture

The Video-processor Design

In response to a video-output signal from the raster display unit the video-processor outputs the current pixel information as it cycles through the mn pixels of the chip. Its access of the processor memory is transparent to the processors. A token that circularly links all the mn pixels of the chip is sufficient, given the simple sequence in which pixel data is accessed for raster display.

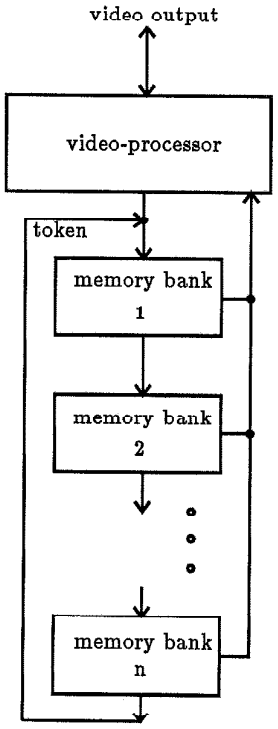


Figure 3.12: Video-processor access of the memory banks

The Memory Design

The total chip memory, corresponding to the pixel and depth buffers for m by n pixels, is divided into n banks, one for each processor. The memory for each processor is independent of the other processor memories.

The processor access of the pixel memory is pseudo-random; between the first and last pixels to be interpolated the pixel memory is accessed sequentially. Thus only the first pixel

access is random. For the subsequent pixels, the address for the pixel just computed can be incremented to access the next pixel to be computed. We implemented this by a token that successively passes through the m pixels of the processor in the spatial order of the pixels in image-space. The "x_{start} control" of the preprocessor controls the start of this token so that by the time the processor is ready to access pixel memory, the first pixel to be interpolated has the token and can send its data to the processor on a pre-charged bus. The n tokens on the chip, one for each of the n processors and its memory bank, is independent of the video token that crosses memory bank boundaries.

In the other dimension the $r + g + b + z$ bits of each pixel memory are cyclic because the processor is bit-serial and needs only a bit of information on each system clock cycle. The implementation approximates a circular shift register for the $r + g + b + z$ bits of each pixel. Memory read-out is thus destructive.

Figure 3.13 illustrates the memory organization.

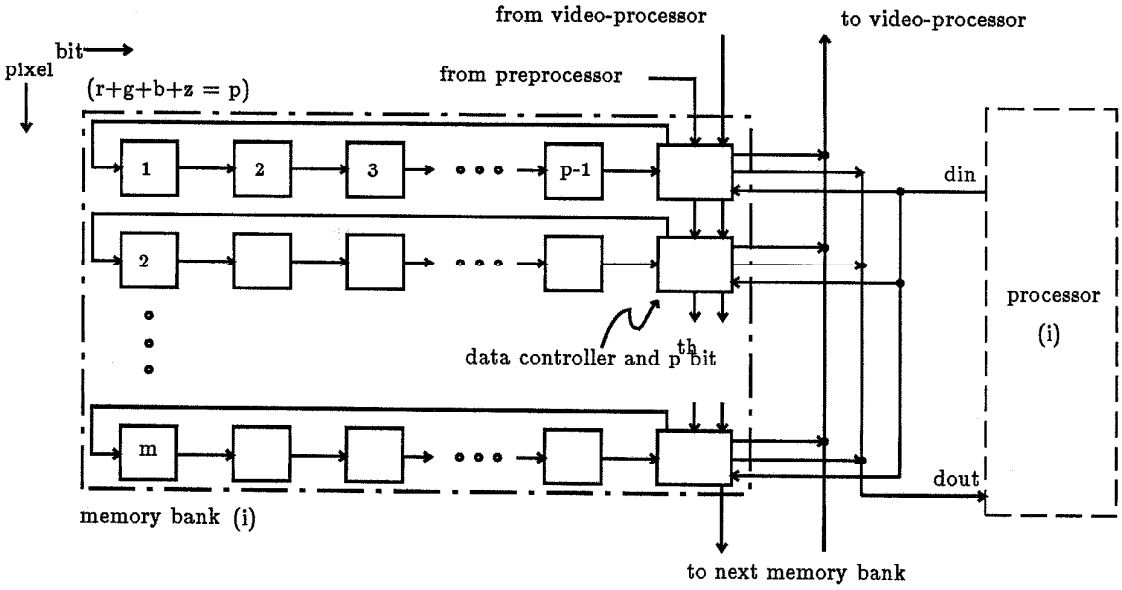


Figure 3.13: Memory Bank Design

Chapter 4

Why this Architecture?

4.1 The VLSI Influence

VLSI has made radical improvements in the speed, power, size, cost, and computational characteristics of systems that we may implement. These advances translate into very complex circuits inside chips – this can be inferred from the acronym VLSI. A more subtle and potentially more significant fallout is the effect on the architecture of systems that we may construct with these chips. VLSI provides an effective implementation medium for architectures and algorithms that can break the von Neumann barriers.

In system architecture the major philosophical change comes from the economic feasibility of having memory and logic in the same technology. A single chip or a small subset of chips can now constitute efficient, self-sufficient processing elements. It is easier to conceive of and apply multiple, concurrent, and local processing elements to create powerful and flexible solutions to problems. With the information supply close to the processing task multiprocessing can exploit the parallelism inherent in certain algorithms without inhibitive memory contention. The physical characteristics and economics of VLSI encourage specialized hardware alternatives to software; and the easing of technological restrictions permits us the freedom of creating innovative architectures in these solutions.

The advances in VLSI architectures require concomitant advances in algorithm formulations and methodologies. The data dependency, concurrency, computational requirement, and communication structure of the algorithm formulation should map the characteristics of the underlying hardware. Conversely, system architectures for memory bound, communication bound, and computation bound algorithms should differ. The best approach is to simultaneously develop the algorithm, or the reformulation of an existing algorithm, and the architecture to implement it.

Within the chip, memories are no longer "passive" – the marriage of logic with memory behooves us to consider them as a composite. We can delegate computation and intelligent

information access to a very fine grain of processor-memory module. This lattitude is however to be tempered with the principle of maximum locality of processing and reference that communication restrictions enforce.

In particular, VLSI favors architectures based on identical modules connected in simple and regular structures. Modular design facilitates layout and repetition of the modules eases testing and reduces design and fabrication costs. Flexibility of application requires extensible designs – a strong argument for homogeneity. Regular and local connections add to extensibility and encourage an algorithm formulation that simplifies communication.

The scarcity of the communication resource profoundly affects all aspects of VLSI architecture and algorithm design. Resolving communication contention is an integral part of architecture design; it is manifest in many characteristics of our design.

4.2 Communication Bottleneck – the von Neumann equivalent in VLSI

VSLI allows us increasing densities of logic and storage on a chip. The primary limitation lies in the interconnections; wires cover most of the chip area. Higher densities exacerbate parasitic effects requiring signal drivers and sensitive sensing circuitry. Communication between chips extracts a price in the area of pad drivers, bonding pads, and sense amps. Package pin limitations, chip yields, and the power dissipated in switching these capacitive loads now become important parameters in the total system equation. Parasitic wiring capacitances introduce delays and a resultant loss in performance. Communication thus strongly affects the power dissipated, cost in terms of silicon area and packaging yield, and the speed of VLSI circuits.

VLSI architectures bear the burden of addressing the communication problem directly and effectively. Since most of the computation expense, time, and energy are expended in communicating over long distances, we strictly adhere to the principle of locality or localized communication. Off-chip communication in critical paths is discouraged. Designs with global communication and/or rippling logic are replaced with designs that only require local and near-neighbour communications.

With an understanding of the system characteristics favored by VLSI, let us follow the design evolution of our scan conversion system. This will make it easier to recognize the pros and cons of current approaches to this application.

4.3 Architecture Evolution of the Scan Conversion System

To start with, the host scan converted all polygons and responded to the video scanning signal for display. This was an inefficient use of the host processing power and led to a

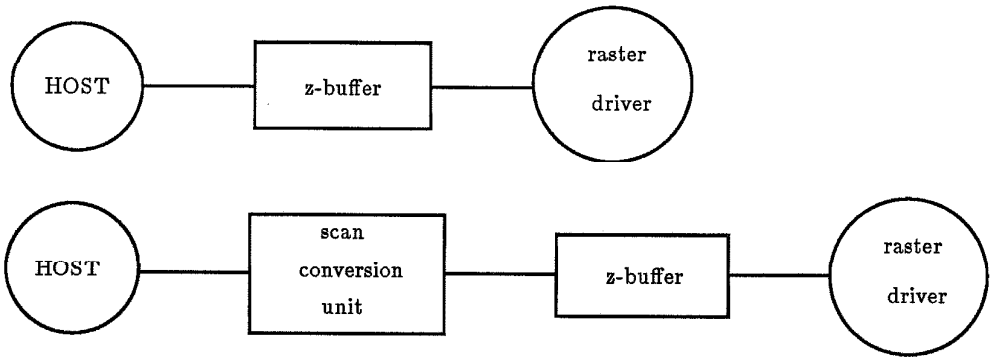


Figure 4.1: The two most primitive architectures

specialized scan conversion unit attached to the host, and an external z-buffer as shown in Figure 4.1. The limited memory bandwidth of this approach and the flexibility of VLSI motivated the move to the distributed system of Figure 4.2. The scan conversion is

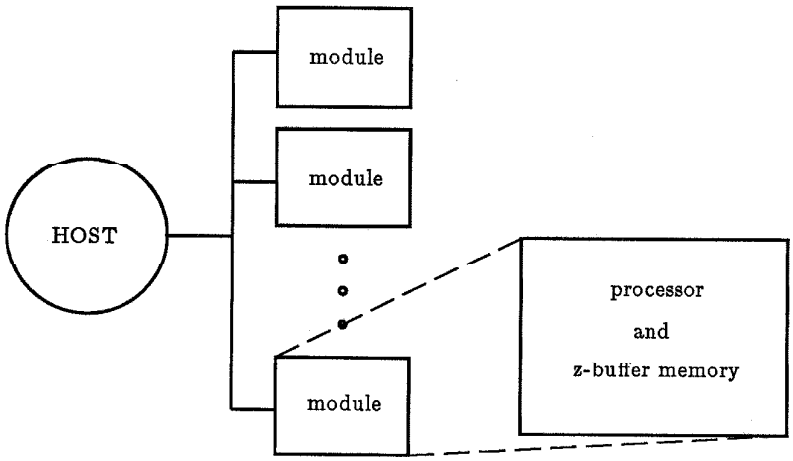


Figure 4.2: The first attempt at a distributed system

now distributed over a number of modules so that each module is responsible for a unique sub-image. A module consists of a processor and the corresponding z-buffer memory. The processor-memory bandwidth is improved by a factor equal to the number of such modules and the increase in throughput reflects the parallelism in the system. We broadcast the same polygon to all the modules simultaneously only after all the modules have finished pro-

cessing the previous polygon. The global communication, however, leads to communication bottlenecks and an increased sensitivity to clock skew. Also, all the modules are functionally distinct. To differentiate the modules from identical implementations we have to add an initialization phase to the processing, requiring an increase in communication, processor logic, and host involvement. Moreover, processor utilization is low and gets lower as we add more modules to the system to increase parallelism. Tesselating and interlacing the image space distributes the work load more equitably, but again increases a combination of communication, preprocessing, and processor complexity. The lock-step-with-polygon nature of the processing restricts the use of pipelining in the scan conversion task. An orthogonal approach to distributed rendering utilizes pipelining as the primary vehicle to increased throughputs. Please see Figure 4.3.

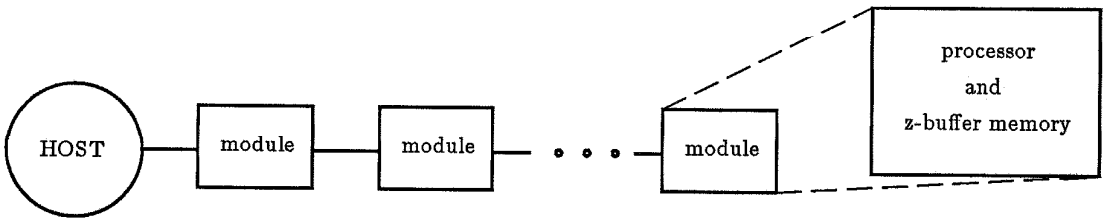


Figure 4.3: The pipelined system

We now have only local communication. The processor utilization is higher at the expense of an increase in latency. This architecture encourages incremental algorithms which have the advantage of simplified preprocessor and pixel arithmetic. Scan line interpolation for different scan lines being independent, a number of such pipelined systems, each responsible for a mutually exclusive set of scan lines can operate independently and in parallel to give us a multifold increase in throughput. At one end of this parallel spectrum we have an independent pipelined system for each scan line. In the pipeline spectrum, the highest throughput corresponds to a processor for each pixel of the scan line. Note that the processor per pixel combination of this architecture has better performance, communication, processor utilization, and system design characteristics than a processor per pixel in the previous architecture with global communication. Also, we can employ a similar incremental approach to preprocessing, *i.e.* the conversion of polygon information to scan line information. Figure 4.4 and Figure 3.6 depict the PS system architecture that resulted from the evolution.

In implementing this scan line system architecture in VLSI, we concentrated on the following issues:

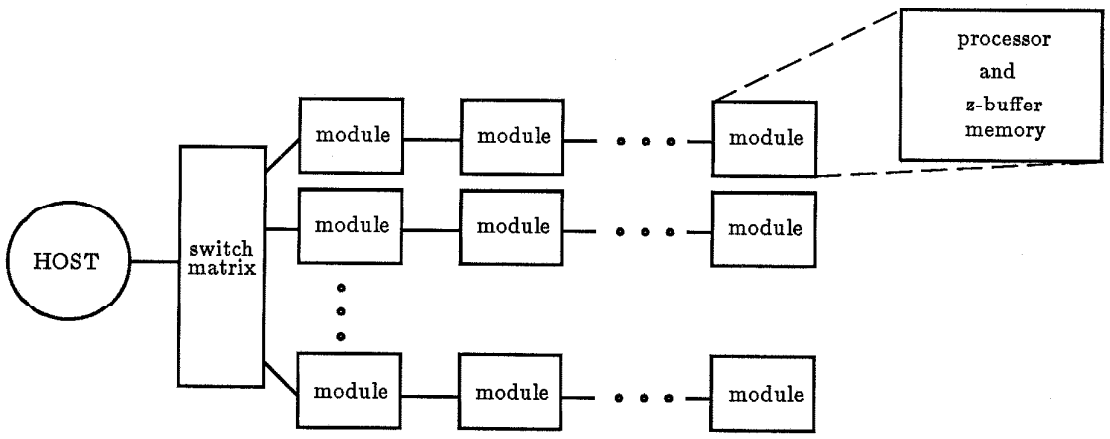


Figure 4.4: The PS Architecture that evolved

- Chip set size
  Minimizing the number of custom VLSI chip types. We have one. And minimizing the numbers of this chip in the system.
- Die size
  Minimizing the size of the chip.
- Communication
  Maximizing the communication bandwidth while adhering to the strictly local communication model.
- Pin count
  A low pin count and a small die size give higher yields in fabrication, an important factor determining the viability of the design.
- Power
  Minimizing the dynamic power dissipated in the chip.
- Speed
  Maximizing the polygon rate. We focussed on maximizing the throughput once the latency was found to be a small fraction of a frame time.

In the context of the above tradeoffs we applied many interrelated philosophies, described in the next section, in the design of the chip.

4.4 Salient Features of the PS Architecture

4.4.1 Bit-Serial Arithmetic

Our architecture is communication limited. Bit-serial arithmetic helps reduce the clock period and thus boosts the communication bandwidth by a factor close to the number of bits of alternate parallel hardware. Serializing memory and logic in the design of a module while increasing the number of parallel modules improves hardware utilization and throughput at the cost of an increase in latency.

Bit-serial processors improve the flexibility of data formats. In our case, altering the data format or resolution implies minor changes in the clock and timing signal generator. It also helps in lowering the pin count and silicon area of the chip. Enhancing the connectivity of the structure, say to a hexagonal connectivity for dithering, now will not increase the pin count inordinately.

4.4.2 Systolic Adaptation

We have the strictly local and regular communication structure of systolic architectures. The DDA we implement, closely maps the pipelining inherent in the systolic approach. Our adaptation diverges from a purely systolic computation in that every processor does not compute on every clock tick. We use the systolic paradigm to capture the concepts of parallelism, pipelining, and interconnection structure but we do not subscribe to the strict lock-step computation of systoles. Just as in systolic systems two communicating processors have a data path between them but in our case the communication is asynchronous.

Another divergence is that the sequencing of operations is neither built into the nodes nor are control signals broadcast into the array in an SIMD fashion; in our adaptation the control for the computational array is distributed into the array. In the current implementation, the data encodes the operation – normalize, interpolate for a variable number of pixels, kill the data packet – mimicking a data-flow architecture. In manipulating the data packet each processor is effectively encoding the command for the next processor. Additionally, the data packet also contains an explicit command field. Currently the only encoded command controls z-compare. PS allows each processor to affect the operation of subsequent processors in the scan line by manipulating the data and also by explicitly manipulating the command field. We hope to exploit this flexibility to address complex applications.

4.4.3 On-chip Memory

Scan line conversion requires a high memory bandwidth. Keeping in mind the penalties that off-chip accesses entail in terms of power, speed, pin count, and silicon area, we have a compelling reason for on-chip memory. There are other not so obvious advantages of on-chip memory. Distributed memories that are in close proximity to the processor offer less memory contention and global address decoding. The on-chip memory and processor share a common clock reducing concerns of clock skew.

We can exploit the VLSI advantage by specializing the on-chip memory to the processing task. In our case this corresponded to a multiple-ported, dynamic memory that mimics a circular shift register for each pixel. It is optimized for bit-serial arithmetic. The data dependencies and locality that are typical to our processing task give us a simplified, fast, and small memory decoder, and reduced wiring.

4.4.4 Algorithm Implementation

The algorithm implementation for the distributed system tracks the simple and local connectivity of the systolic structure. The data structure carries all relevant information thus avoiding the need for common busses or global communication. It automatically tesselates polygons onto the underlying chip array structure adding extensibility to the implementation. Tesselation does not incur any space or time overhead as would be the case in a higher level partitioning scheme.

4.4.5 Processor-Pixel Distribution

We want the processor-pixel distribution that minimizes communication and connectivity requirements, and distributes the workload most equitably to maximize parallelism. There are two aspects to the distribution; the first applies to an optimal choice of m, the number of pixels per processor, and n, the number of processors per chip. The second deals with the best mapping of these m pixels to each processor.

The m vs. n consideration comes from our attempt to have PS exhibit parallelism that is closest to a processor per pixel approach. Resource utilization, communication bandwidth, and VLSI implementation criteria make a processor per m-pixel and n-processor per chip design more optimal. Increasing m reduces the concurrency, eases communication requirements, improves processor utilization, and requires fewer processors for a specified display size. With $m = 1$ we have a processor per pixel and the best possible concurrency in operation. Increasing n increases the communication requirements per chip, but requires fewer chips for a specified display size.

The m pixels per processor may be arranged as a sub-image of size u pixels by v pixels, where $uv = m$. For a constant m, increasing u reduces pipelining and eases communication

requirements; increasing v helps pipelining but exacerbates communication requirements. Since communication is the bottleneck in PS, we chose $u = m$ and $v = 1$, the combination with the highest communication bandwidth. This combination allowed us to exploit some known data dependencies that resulted in simple and elegant structures. For example, interpolation for pixel $p + 1$ of a polygon follows that for pixel p on the same scan line and uses the same data packet as data packets carry information for one scan line and one polygon. We used this pixel coherence to design the simple token decoder. Interpolation for pixel $p+1$ uses the same incremental values as for pixel p from local and fast registers instead of reassembling the modified data packet, sending it to the next processor, disassembling the data packet there, and decoding for pixel $p + 1$. Also, locality of reference suggests that a data packet for scan line $i + 1$ will follow that for scan line i. A large v increases the probability that consecutive data packets are to be processed by the same processor and thus increases the probability of blocking the pipeline.

In the current implementation the number of clock cycles to transfer a data packet is equal to the number of clock cycles to interpolate for a pixel. Thus with $u = m$ and $v = 1$ the communication and processing requirements for a chip are most optimally matched for $m = n$.

4.4.6 Hierarchical and Distributed Control

The primary considerations in the design of the control circuit are the silicon area, the speed of the chip, and the design time. For VLSI these are direct manifestations of the cost of the circuit. We employed a hierarchical design that separates the sequencing and the generation of commands. [Obrebska]

In our design the clock and timing signal generator forms the top level of the hierarchy of the control. By extracting the timing signals that are common to the different functional units of the chip we avoid repetition and redundancy in the signal generation. These signals form the skeleton of the control structure providing validation and supervision for the flesh of the structure within and between the units. This flesh is distributed at various levels. The distribution gives us flexible and localized control, and minimizes long-distance communication alleviating the communication malady discussed in Section 4.2.

The clock and timing signal generator is implemented as a PLA. The PLA construct is easy to generate, test, and debug. Its use at this level makes the implementation flexible. We can change the parameters of the implementation *viz.* the system vs. processor clock ratio, the data format, the buffer sizes, the screen resolution, the number of pixels per processor (m), with minor modifications. The versatility of the PLA structure, the irregular nature of the control logic, the large size of the corresponding FSM, and the fact that there is only one of these per chip justify the area of the PLA.

Token-passing is used to control communications between units. Pre, post and processor

communications use one polling token, video communication employs a token that circulates through all the m by n pixels of the chip, and each processor-memory bank communication is controlled by an independent token; thus each chip has $n + 2$ independent tokens. All information transfers from the processors to the postprocessor, from the preprocessor to the processors, and between each processor and its pixel memory bank are initiated by the processor clock. A processor clock period being $r + g + b + z$ times the system clock period, we have $r + g + b + z$ system clock cycles for decoding communication requests. At the same time, we want to avoid personalized control lines for every type of communication. Token-passing gives us the best tradeoff between decoding time, size of decoding circuitry, and multiplicity of control lines. The use of a shifting token for processor-memory bank control exploits pixel coherence (interpolation for pixel $i + 1$ follows that for pixel i). The video-memory token requires no decoding.

Within each unit, control is implemented by gates and dynamic shift registers mimicking the "delay-element method". [Hayes] This design style conserves area and reduces delays because of the small incremental cost of latches in MOS technologies. The internal functions of the processors are directly controlled at this level. It is therefore repeated as many times as the number of processors per chip (n); this makes the area considerations significant. The regularity of the algorithm and the "tight" properties of this design style justify its non-uniformity and thus lack of automation.

4.4.7 Logic Implementation

The use of steering logic allowed us to exploit the features typical of the implementation technology, namely CMOS. We have pass transistor logic networks with the logic level appropriately restored. These networks control data or signal flow between data latches – another elegant and low cost feature of MOS logic.

Static CMOS logic, however, requires us to implement both the complimented and uncomplimented forms of the function in n-type and p-type transistors respectively. Not only is silicon area wasted in the duplication, but the problem of routing signals is aggravated. We used precharge logic whenever possible. Besides the savings in circuitry and routing, precharge logic gives us lower capacitances and hence higher speeds. With only one p-type transistor in the pull-up path, load capacitances charge faster. We easily derived mutually exclusive compute-precharge phases from the two-phase non-overlapping clock on the chip thus fulfilling the precharge requisite that inputs to precharged busses be glitch-free and that the bus not be used for the time it needs to precharge. We opted not to concern ourself with the lower speed bound in using dynamic logic.

4.5 Comparison with Existing Polygon Rendering Systems

Our design exploits parallelism by partitioning the image space so that each processor is responsible for only a sub-image and consequently fewer objects. Another possible approach is to partition the object space so that each processor is responsible for only a subset of the objects in the image. We compare the PS architecture with three prominent and recent architectures for scan conversion that apply image space partitioning. These are:

- Pixel-planes by Henry Fuchs et al. at the University of North Carolina
  [Fuchs 83,Fuchs 81,Poulton et al. 85]
  Pixel-planes, in our knowledge, is the only processor-per-pixel architecture developed to Gouraud shade and depth-buffer polygons. The designers are now studying other applications for the architecture. The most unique aspect of the architecture is the use of multiplier trees to calculate the linear equation, $F(x) = Ax + By + C$, at each pixel. Differently said, the multiplier trees decode linear equations onto the pixel space.

- Scan Line Access Memories by Stefan Demetrescu at Stanford University
  [Demetrescu 85]
  Scan Line Access Memories, or SLAMs, use conventional high density RAMs enhanced with limited processing for high speed rasterizing of large sections of scan lines. The current implementation runlength encodes a specified 16-bit halftone pattern to affect an entire, or a subset of, a scan line. The architecture draws strength from its simplicity and the use of conventional RAM design concepts.

- Super-Buffers by Nader Gharachorloo et al. at Cornell University
  [Gharachorloo 85]
  The Super-Buffer architecture is a systolic approach to the task of rasterizing polygons. Super-Buffers, developed independently, share many features with PS. Notably, we both divide the task of rendering polygons into polygon edge interpolation and scan line interpolation. In polygon edge interpolation both systems employ an incremental algorithm. In addressing the communication problem, our approach is a variation of the systole to which the Super-Buffers adhere strictly.

In our comparison, we characterize the efficiency of an architecture by:

- the extensibility of the architecture to increasing functionality.
- the extent of host involvement.
- the speed potential.

- the processor utilization.
- the communication bandwidth.

Only PS and the Pixel-Planes currently implement Gouraud shading and depth-buffering. SLAMs and Super-Buffers have a one-bit plane at each pixel and do not support hidden surface removal.

Pixel-Planes inherently support all functions that can be represented as a linear equation. Functions that do not have the linear equation as their natural representation require extensive preprocessing. For example, polygons are most conveniently described in terms of their vertices. Edge equations are easily obtained from this information; however planar equations for color require more preprocessing. Super-Buffers are limited in that no memory is retained past scan line boundaries, requiring all pixel information to be retransmitted through the row of Graphics Engines at the onset of scan lines. This requires pixel information for the entire image space to be stored externally along with data for all the active polygons. The SLAM architecture is optimally designed for a 16-bit function repeatedly applied to all pixels on a scan line. The architecture loses most of its elegance and performance metrics for more involved processing. The PS architecture is optimized for incremental arithmetic and local communication. With the understanding that all global communication can be implemented as local communication using systolic conversions, we foresee few limitations to applications with our architecture.

Global communication in Pixel-Planes and SLAMs limit the performance potential of the systems. In Super-Buffers, the image clock is also the video clock limiting the polygon throughput. In all of these systems the memory is essentially single-ported; thus, video access and image processing are not completely transparent. Our simple decoding structure gives us multi-ported on-chip memory with completely independent and transparent video and image accesses. Also, in all the other architectures that we have considered all the chips are functionally different. This translates to additional logic, delay, and host involvement in an initializing phase. With a simple "normalizing" function, we avoided that requirement.

Processor utilization in Pixel-Planes is low by virtue of the processor per pixel approach to a lock-step-with-polygon character enforced by the global communication. In SLAMs, processor utilization is high only for applications with a repeatedly applied pattern covering large sections of a scan line. Both Super-Buffers and PS enjoy a high processor utilization profile. In the Super-Buffers however, blank scan line commands for flushing the data out of the processor row limit processor utilization.

Chapter 5

Quantitative Analysis of the PS System Performance

Since all processing in the chip is bit-serial and we have multiple bits of information per pixel to represent the attributes z, r, g, b, the interpolation for each pixel takes multiple clock cycles. Multiple clock cycles are also required to transfer a data packet between chips over the limited number of pins. In our implementation these two delay times are equal. Given this constraint, the discussion on processor-pixel distribution in the previous chapter showed that m, the number of pixels per processor, is optimally equal to n, the number of processors per chip. Therefore our parameters are:

- p, the number of clock cycles to transfer a data packet between chips and also the number of clock cycles to interpolate one pixel, (units: pixel^{-1}),
- c, the clock speed, (units: second^{-1}),
- X, horizontal screen size, (units: none),
- Y, vertical screen size, (units: none),
- N, the number of processors per chip and also the number of pixels per processor, ($N = n = m$), (units: none), and
- v, the number of clock cycles to transfer information for one pixel in response to the video scan signal, (units: pixel^{-1}).

Thus we have a two-dimensional processor array of size $ceil(X/N)$ by Y, and a two-dimensional chip array of size $ceil(X/N)$ by $ceil(Y/N)$ for a screen of size X by Y. Let us simplify the calculation by assuming that N divides X and Y. With X/N processors per scan line and N scan lines per row of chips, we have X processors per row of chips.

5.1 Pixel Rate

The pixel rate is the number of pixels processed by the system per second. (units: pixels/sec)

5.1.1 Best:

The best pixel rate corresponds to all the processors in all the chips processing pixels in parallel. Note that there are XY/N processors. With p clock cycles to interpolate a pixel and c clock cycles per second,

$$\text{the best pixel rate} = cXY/(pN) \text{ pixels/sec.}$$

If data packets were to affect entire scan lines then the communication is symmetrical and we have a constant rate of data packet transfer between every two consecutive processors on the scan line. Thus the communication bandwidth at the first processor applies to the entire scan line. A processor on the chip can interpolate for a maximum of N pixels. Thus a processor will require a new data packet only after N pixel interpolate times. But with the pixel interpolate time the same as the data packet transfer time, we can transfer data packets to the other $N-1$ processors on the chip between data packets for the processor in consideration. Thus the processors and the communication pins are always busy and the best pixel rate corresponds to the maximum communicating and processing bandwidths.

Note that the two extreme cases for data packets that affect entire scan lines are screen size polygons where all data packets affect entire scan lines, and Y different polygons each spanning an entire scan line.

5.1.2 Worst:

The worst pixel rate corresponds to one pixel repeatedly painted. In this scenario only one processor of one chip is active at any given time. With p cycles to interpolate a pixel and c clock cycles per second,

$$\text{the worst pixel rate} = c/p \text{ pixels/sec.}$$

Note that in this worst case neither the processor waits for the data packet nor the communication channel for the processor; processing matches communication. The same worst case can also be reached if we were to be either only communication or only processing limited. The former corresponds to repeatedly painting one pixel of each processor of one chip in the array. Now a processor has to wait N data packet transfer times before it can get another data packet and in this time the other $N-1$ processors on the chip will interpolate for one pixel each. The latter corresponds to repeatedly painting all the N pixels of one processor on one chip in the array. A processor can now accept a waiting data packet only after N interpolate times for interpolating the N pixels of the previous data packet.

5.2 Polygon Rate

The polygon rate is the number of polygons processed by the system per second.
(units: polygons/sec, where a polygon unit is expressed as a number of pixel units)

5.2.1 Best:

In computing the best polygon rate we would like all the processors to be working on different polygons. With XY/N different processors this requires that many different data packets in the system. The asymmetry of communication in PS implies that we need to communicate to the chips in the first column of the array, data packets for their own processors as well as data packets for all the chips in the rest of the row. Thus the best polygon rate can be determined from the maximum communication bandwidth of the first column in the array. With p clock cycles for communicating one data packet and c clock cycles per second, we can transfer a maximum of c/p data packets to the first chip of each row of chips. However if a polygon only spans one scan line, each data packet corresponds to a distinct polygon and we can transfer c/p polygons per row of chips. With Y/N such rows,

$$\text{the best polygon rate} = cY/(pN) \text{ polygons/sec.}$$

Note that the best polygon rate is limited primarily by the communication bandwidth. If a data packet transfer took one clock cycle with processing still at p, then we could continually transfer up to p data packets to p different processors in a row of chips. If a data packet transfer took only one half a clock cycle then we could transfer up to $2p$ data packets to $2p$ different processors without wasting any processing time of these $2p$ processors. With X processors in the row and p clock cycles for interpolating one pixel, the processing bandwidth would limit the best polygon rate only if the communication bandwidth increased beyond X/p data packet transfers per clock cycle. If each data packet corresponded to a different polygon, with X/p data packet transfers per clock cycle we could render a maximum of X/p polygons per row of chips per clock cycle. These polygons would be one pixel per processor polygons and the following scenario is an example of the best case behaviour. A processor gets a data packet. The processor takes at least p clock cycles, the time to interpolate for a single pixel, to process this packet(polygon). In these p clock cycles, we transfer data packets(polygons) to the other $X-1$ processors in the row of chips. When this processor completes interpolating we are ready to transfer a new data packet to it.

With Y/N rows of chips and c clock cycles per second the best polygon rate of the system would be $cXY/(pN)$ which is the best pixel rate. This underlines the asymmetry of communication within the system.

5.2.2 Worst:

The worst polygon rate can result from processing limitations, communication limitations, or both. Repeatedly painting one pixel per processor for one chip in the array gave us the worst pixel rate, c/p, due to communication limitations. Now if all these N pixels belonged to the same polygon we get,

$$\text{the worst polygon rate} = c/(pN) \text{ polygons/sec.}$$

This corresponds to a polygon with at least one pixel per scan line for at least all the scan lines of one chip. Processing limitations gave the worst pixel rate by repeatedly painting all the N pixels of one processor on one chip in the array. If all these N pixels belonged to the same polygon we get the same worst polygon rate as above. This corresponds to a polygon with at least one scan line in which it spans all the pixels of that scan line for one chip in the array. We reach both the processing and communication limits if we repeatedly paint a polygon that spans all the N^2 pixels of at least one chip in the array. One example of such a polygon is a screen-sized polygon. A screen-sized polygon has a pixel rate of $cXY/(pN)$ and XY pixels. Therefore the worst polygon rate also corresponds to repeatedly painting screen-sized polygons.

5.3 Frame Rate

The frame rate is the number of frames that may be accessed from the system per second. (units: frames/sec, where a frame unit is expressed as a number of pixel units.
i.e. one frame = XY pixels)

The best frame rate requires that each chip has a unique data bus to the D/A convertor. In that case we require v clock cycles to output the information for one of the N^2 pixels on the chip. Thus for all the N^2 pixels, we require vN^2 clock cycles and assuming the video rate is also the clock rate, c,

$$\text{the best frame rate} = c/(vN^2) \text{ frames/sec.}$$

This approach is unreasonable in the amount of external wiring. A more reasonable approach will have one pixel bus for a row/column of chips and a common video clock to all the chips on a column/row. With the row pixel-bus, we can access one of XN pixels per v clock cycles. This gives

$$\text{the row pixel-bus frame rate} = c/(vXN) \text{ frames/sec.}$$

With the column-pixel bus, we can access one of YN pixels per v clock cycles. Thus

$$\text{the column pixel-bus frame rate} = c/(vYN) \text{ frames/sec.}$$

5.4 Latency

The latency is the delay between the time that a data packet is placed at the input port of the system and the time that interpolated pixel data for one of the data packet pixels may be read at the output port. (units: seconds/pixel, but more commonly expressed in seconds as the definition assumes that the calculated value is for the first pixel)

The delay in latency calculations includes the communication delay in getting the data packet to the destined processor, the computation delay for the interpolation, and the delay incurred in accessing the video bus. Thus the latency is dependent on the video bus structure. We shall only consider the worst case latency.

The maximum data packet communication delay corresponds to a data packet destined for the last processor of the row. With X/N processors in a row and p clock cycles to transfer a data packet through one processor, the delay incurred is Xp/N clock cycles. The computation delay is the p clock cycles to interpolate for a pixel in the data packet. The row pixel-bus cycles through the XN pixels in a row. Therefore with v clock cycles for outputting data for one pixel, the maximum delay in accessing the video bus is XNv. Again, assuming the video access rate is also c clock cycles per second,

$$\text{the worst case row pixel-bus latency} = (Xp/N + p + XNv)/c \text{ seconds.}$$

The worst case column pixel-bus latency can be similarly calculated.

There are two interesting observations:

- the best case performances improve with increasing screen sizes while the worst case performances remain unaffected,
- the communication bandwidth matches the processing bandwidth for the worst pixel and polygon processing rates. However increasing the communication bandwidth by a factor up to X increases the best polygon rate by the same factor. At the maximum processing bandwidth, the factor X results from the asymmetry in the communication as the first chip of the row has to process/transfer data packets for all the X processors in the row.

Chapter 6

Application Possibilities and Enhancements of the PS Architecture

This chapter is organized in an extensibility precedence: the extensions to the system are described in a decreasing order of detail and/or an increasing magnitude of modifications. All the enhancements share the same design philosophy – on-chip memory, decoding that exploits data dependencies, local communication, bit-serial processing – as the current system, and most cases propose a simple increase in the functionality of the processor.

6.1 A Better Processor-Pixel Distribution

The connectivity of the chip array in the current implementation introduces an asymmetry in communication. The communication requirements decrease as we move from the first column of chips to the last as the first chip in the row has to transfer/process data packets for all the chips in the row. As we move down the row, in increasing x, data packets terminate and the rate of data flow between chips falls.

In the processor-pixel considerations, we showed $u = m, v = 1$, and $m = n$ to be most optimal for each chip. Currently, the n processors of a chip are stacked vertically so that each chip is responsible for a square array of pixels. However, if we connect the processors back-to-back so that they span contiguous sections of the same scan line, as shown in Figure 6.1, we continue to have XY/N^2 chips in the system but we change the aspect ratio of the array. The chip array changes to X/N^2 by Y from X/N by Y/N, and the processor array is the same at X/N by Y. However with X/N processors per scan line and only one scan line per row, we now have X/N processors per row of chips and not X processors per row as

earlier. Also, data packets span potentially larger sections of a chip's domain. These reduce the data packet requirements per row of chips. Conversely, with a chip now processing larger scan sub-sections the data traffic between chips falls. Thus we help remedy the asymmetry of communication discussed above.

The lower communication requirements at the same communication bandwidth improves throughput and processor utilization. In the context of locality of reference which has data for scan line $i+1$ following that for scan line i, we increase parallelism by assigning one scan line per row of chips. Note that we have the same number of processors per scan line leaving the performance contribution of pipelining intact. In this approach consecutive

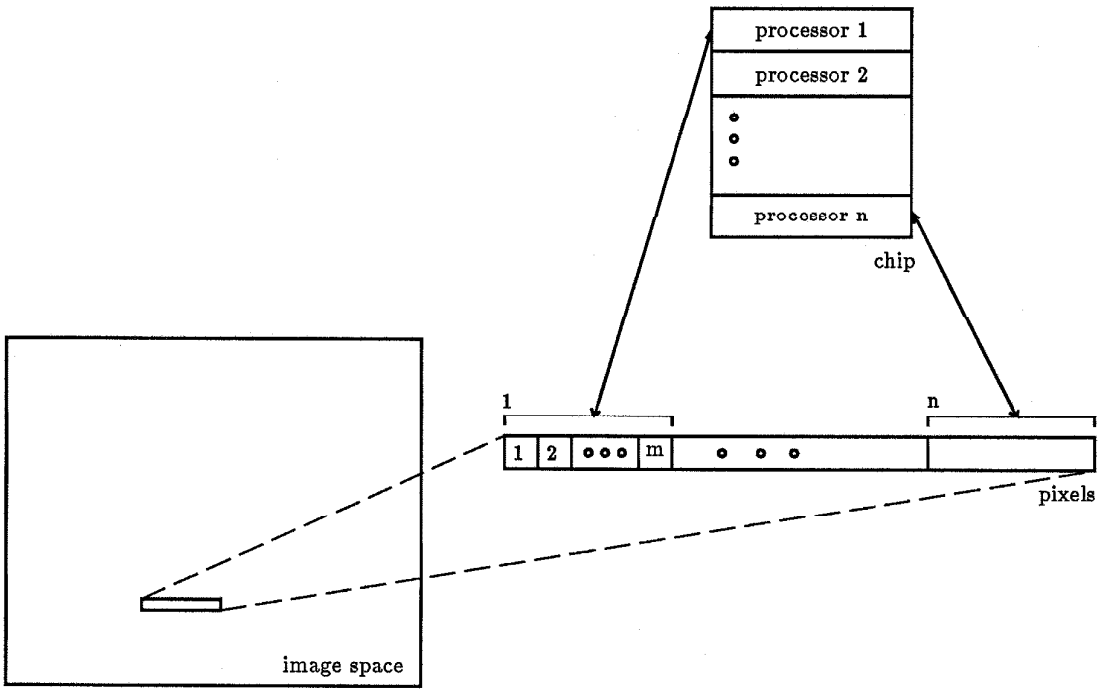


Figure 6.1: The proposed processor-pixel distribution

processors span contiguous sections of the pipeline. In the context of pixel coherence, the data dependency that applies to adjacent pixels within a scan line, we can now simplify and enhance the power of the token decoders for higher performances.

The performance analysis for this modified distribution is interesting. The best pixel rate continues to have all the processors interpolating in parallel and is unchanged. The

worst pixel rate is also unchanged for one pixel continuously painted. The modified architecture exploits data dependencies and since the pixel rate computations consider either unrelated pixels or depend on the total number of processors which is constant, we have no improvement in the pixel rates.

The polygon rates, which do depend on data dependencies, exhibit the communication improvement of the modified architecture. The worst polygon rate is the same as before but now we are mostly processing and not communication limited. The communication limited case of a polygon with one pixel per processor of a chip that corresponded to a thin polygon does not apply anymore. The worst case of a polygon that covers all the pixels of a chip and that was both communication and processing limited is now only processing limited as all the N^2 pixels of a chip correspond to one data packet. The processing and communication bandwidths match each other solely in the screen-sized polygon case. For computing the best polygon rate we have, as before, a polygon that spans only one scan line; each data packet corresponds to a unique polygon and we can transfer c/p polygons per row of chips. However, we now have Y rows and the best polygon rate is cY/p, an improvement of a factor of N. Now we have X/N processors in a row of chips and the communication bandwidth has to improve by a factor of only $X/(Np)$, and not X/p, to achieve the best possible polygon rate that is the best pixel rate.

With the multi-ported memory, PS has the potential for processors that interpolate any of the N^2 pixels of the chip. All the pixels of the chip would then be one bank and not multiple, distinct banks as in the current design. For this memory bank we would then need only one decoder that controlled multiple tokens; the number of tokens would continue to equal the number of processors on the chip. This would add flexibility to the architecture and improve processor utilization and throughput. A new set of parameters would then determine the most optimal number of processors and pixels per chip.

6.2 Processor Enhancements

We only consider applications that can take advantage of the system architecture. We expect that many algorithms could be reformulated to exploit the distributed control, communication and processing structure of the system. The list of possible applications is not purported to be complete. The last two in the list below embody the structure which promises interesting solutions to difficult problems.

6.2.1 Anti-aliasing

Aliasing effects arise from the finite size of pixels. Differently said, it is the quantization error of converting real pixel values to the integer precision of the display. In the DDA computation although the data packet carries only integer information, all the precision of

the real number interpolation at each pixel can be reconstructed from the values in the data packet. As an example, p_{sum}/dy, the difference in the integer x pixel value we adopt and the true x value at the scan line, is available as the integer values p_{sum} and dy. We can use this information to color edge pixels so that edges appear anti-aliased. Note that this is only an approximation to true anti-aliasing and hence is not free from defects.

6.2.2 Selective Masks or Bit Patterns

The architecture supports the convolving of scan lines to specified masks or patterns. We have the flexibility of a mask field per pixel per data packet; this implies that for the same polygon different scan lines could be convolved with different color patterns and the patterns for different polygons can be different. The extension to the system to support this feature involves a variable length data packet that could at any time span multiple processors in the row. Then each data packet could carry its mask, and a processor would strip from the data packet the mask section that corresponded to the pixels in its domain. Command bits would encode the convolution operation. This would allow us to selectively manipulate the color map.

One simple application is the clipping of polygons to arbitrary windows. The data packet could either carry the clipping end points on the scan line, there may be multiple pairs of end points, or an explicit mask for all the pixels on the scan line. Each processor would then AND the interpolated pixel value with the mask bits for the pixel.

We predict exciting effects such as objects that blend into and emerge from other objects. Also, a variable length data packet would allow us to display arbitrary images by explicitly sending pixel data instead of scan line data.

6.2.3 BitBlt

By BitBlt [Foley 84] here we mean copying or moving arbitrary sections of the screen to another location on the screen with one command.

An extension of the variable length data packet concept would allow us to perform BitBlt operations at very high speeds. The pixel data now originates from the source sub-screen and is routed to the destination sub-screen through the switch network between the polygon edge and scan line interpolator systems. A data packet with the source pixel location, length on the scan line, and destination scan line and pixel location moves through the scan line and affected pixel data is appended to the data stream at the source and stripped from the top of the stream at the destination.

6.2.4 Other Incremental Algorithms

The architecture supports algorithms that can be described incrementally. Bresenham's Circle algorithm [Foley 84] is a typical example. It ideally maps the architecture and can be implemented with only minor enhancements of the processor, making the circle throughputs of the system comparable to the polygon throughputs. Other interesting classes of algorithms and graphical effects can be obtained from the current data structure. A case in point involves successively changing the constants in the data packet. By left-shifting the quotient at each computation, we can trivially compute the function 2^x where x is the pixel attribute. Algorithms that can optimally utilize the parallelism of PS can be found in many (most ?) disciplines. These algorithms run inhibitively slowly on current mainframes and the performance improvement by running them on PS would make them practical and useful.

6.2.5 "Adaptive" Algorithms

In our definition every computation on a data stream can be determined by the previous computation in two ways: implicitly by manipulating the data, and explicitly by specifying the command to be performed on the data. An algorithm is "adaptive" if it can be described in a pipeline model that can be changed on the fly. Each data packet can redefine the arithmetic of the pipeline units although the connectivity dictates the pipeline structure. Note that although all the chips are identical they have a programmable flexibility to perform different operations on the data and the command for the rest of the processors in the pipeline. The "program" is carried by the data packet and hence can be altered by the processors.

6.3 Fault-Tolerance

We can incorporate a measure of fault-tolerance by having an extra processor and memory bank per chip and storing the bank attribute (good/bad) in the token decoder. If the processor or memory of any one of the n banks of a chip, say p, is found to be defective the token decoder delays by one clock cycle the valid signal for banks that follow the defective bank in the token path. In this simple way banks $p \ldots n-1$ are effectively replaced by banks $p+1 \ldots n$, where banks $0 \ldots n-1$ are the chip's n banks and bank$n$ is the extra bank added to tolerate a faulty bank.

The detection of a problem is simplified by the existence of the postprocessor; since all banks are identical the same scan line can be sent to successive processors and the result compared in the postprocessor. We do require some additional logic on the chip to do this and to reflect the result in the token decoder. The loss in area incurred by this fault-tolerant

measure is less than $1/m$ times the chip area. Also, the chip forms a generic PS stage and we can replace a faulty chip with a new chip which will reset itself with the background polygon at the start of the next frame. We are not required to interrupt and reinitialize the entire system.

6.4 Considerations in the Design of the Polygon Edge Interpolator (PEI)

The PEI interpolates for the x coordinate as well as r, g, b, and z of the polygon edge at each successive scan line given the data for the two vertices of the edge. The interpolation being linear, the same DDA as the scan line interpolator may be employed. The PEI may be designed to subdivide the object space giving a processor per polygon or the image space giving a processor per subimage as shown in Figure 6.2, or a combination of the two. Typically the subimage of the processor per subimage approach consists of a number of contiguous scan lines to minimize communication. This is similar to the SLI processor spanning contiguous pixels on a scan line. The difference with the scan line interpolator structure, which also partitions the image space but which has a processor span only a sub-scan line, arises from the x and y coherence of a polygon edge as opposed to the x coherence alone of a scan line.

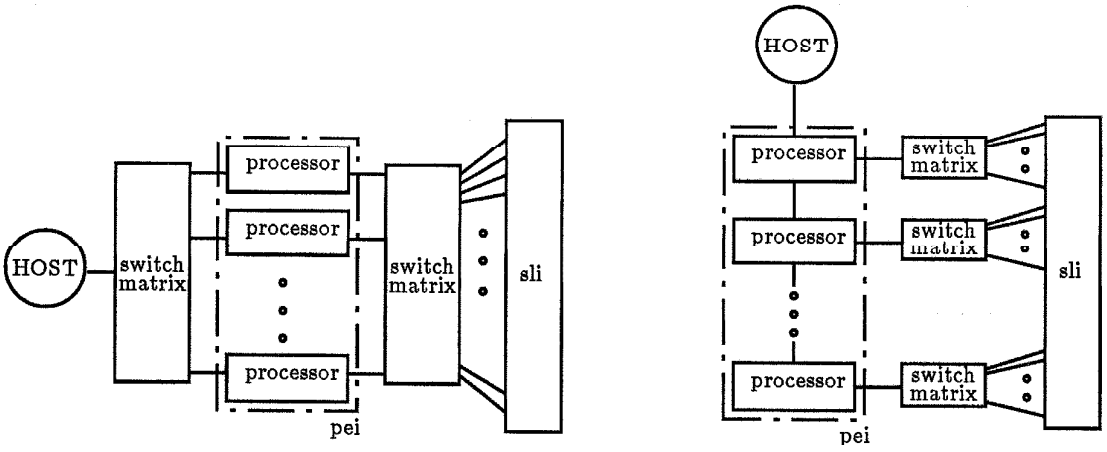


Figure 6.2: Possible approaches in the PEI architecture

From Figure 6.2 we can see that the processor per polygon requires a larger multi-input switch matrix as the polygon may potentially span all the scan lines, while the processor per subimage requires a number of smaller single-input switch matrices. The processor per polygon requires more host involvement as the host has to monitor the processors in order to send edge information for the polygon current to the particular processor, or for a new polygon, in response to requests from multiple processors. Also the host output to the processors and the processor output to the host need a mux/demux. In the processor per polygon approach, the large host window into the working of the system entails broadcast-like communication which we want to avoid.

In the scan line case data packets were not ordered as they belonged to either different scan lines and hence were independent, or to different polygons. In the case of the PEI, all edges have to be ordered within, and between polygons. Thus for the processor per subimage the polygon edge data packet needs a polygon identification field and a sequencing field for a given polygon *id*. The order that polygon edge packets are input to the system and the buffer capacity of the PEI for storing edge information will strongly affect the throughput and utilization of the PEI.

6.5 Architecture Enhancements

With a major redesign effort, but maintaining the design philosophy outlined at the onset of this chapter, we could enhance the connectivity and processing power of the system. Each chip is now additionally connected to the chips directly above and below it in the array. Since all processing is bit-serial, the computation, communication, pin count, and storage requirements are still manageable. With access to the information of nearby pixels, other functions such as dithering can be implemented. The processor array will now render vectors without having to treat them as polygons and having to create multiple scan line commands. The array will also directly interpolate along polygon edges and the processors on the left edge will create scan line data packets for the rest of the processors on the scan line. This will alleviate some of the asymmetries of our communication structure. We could also make the communications bidirectional, as opposed to the strictly unidirectional left-to-right communication of the current implementation, for a truly general system. With bidirectional communication, we could implement the network as a toroidal mesh and introduce the data packets at the center of the image to further reduce the asymmetry of communication.

Chapter 7

Conclusions

PS exhibits the characteristics of distributed systems: increased performance, extensibility, and modular architecture. An eclectic bag of architectural methodologies directly addresses the limitations on communication bandwidth and processor utilization. The power of PS lies in the extensive pipelining and parallelism it incorporates. Bit-serial arithmetic, systolic connectivity, on-chip memory, and hierarchical control effect the performance metrics and extensibility of PS. In its implementation, PS exploits the fabrication technology, CMOS.

A number of tradeoffs and marriages make the PS architecture unique: the distributed communication of systoles leading to high bandwidth and the independence of asynchronous communication; decoding time *vs* the minimal control wires as in token-passing; the low processor utilization of a processor per pixel *vs* the communication and processing bottlenecks of an m by n approach.

In the context of the current application, the reformulation of the DDA maps the architecture efficiently. A judicious processor-pixel distribution is shown to enhance the symbiotic relationship of the application and the architecture. The proposed architecture exploits the data dependencies peculiar to the task of scan conversion. The exercise underlines the advantages of concurrently developing the algorithm, data structure, and architecture.

In designing the PS system, we witnessed an anachronism in design psychology: "memory" = "RAM". Even for the on-chip memory of specialized hardware, the organization is or closely resembles a RAM. However, in most cases, certainly in ours, random access is overkill; And there is a price to pay for it! For most applications, data coherencies can lead to faster, smaller, and more efficient memories. Semi-random access memories are finer tuned to the application and therefore more effective. A case in point are the *multiple ports* of the PS on-chip memory.

Appendix A

Implementation Details

A.1 Architecture Details

m, the number of pixels per processor = 16
n, the number of processors per chip = 16

Pixel Data Format:
r, g, b = 8 bits each
z = 16 bits
Therefore the depth buffer stores 40 bits per pixel. This corresponds to:
40×16 = 640 bits per memory bank, and
$16 \times 40 \times 16$ = 10,240 bits of memory per chip.

1 processor cycle = 48 system clock cycles.
The chip was designed to run at clock rates of 40 Mhz and over.

A.2 System Performance Numbers

$p = 48$ clock cycles per pixel
$c = 40$ MHz
$X = 2^{10}$ this allows a screen pixel to correspond to a
$Y = 2^{10}$ 1 X 1 array of image or virtual pixels
$N = 16$
$v = 6$ clock cycles per pixel

the best pixel rate $= cXY/(pN)$ pixels/sec.
$\approx 5.4613 \times 10^{10}$ pixels/sec.

the worst pixel rate $= c/p$ pixels/sec.
$\approx 833,333$ pixels/sec.

the best polygon rate (current architecture) $= cY/(pN)$ polygons/sec.
$\approx 5.3 \times 10^{7}$ polygons/sec.

the best polygon rate (modified architecture) $= cY/p$ polygons/sec.
$\approx 8.533 \times 10^{8}$ polygons/sec.

the worst polygon rate $= c/(pN)$ polygons/sec
$\approx 52,083$ polygons/sec.

the best frame rate $= c/(vN^2)$ frames/sec.
$\approx 26,041$ frames/sec.

the row pixel-bus frame rate $= c/(vXN)$ frames/sec.
≈ 407 frames/sec.

the column pixel-bus frame rate $= c/(vYN)$ frames/sec.
≈ 407 frames/sec.

the worst case row pixel-bus latency (current) $= (Xp/N + p + XNv)/c$ seconds.
$\approx 2.5356 \times 10^{-3}$ seconds.

the worst case row pixel-bus latency (modified) $= (Xp/N^2 + p + Xv)/c$ seconds.
$\approx 1.596 \times 10^{-4}$ seconds.

Technology: 3 micron cmos/bulk
Silicon Vendor: MOSIS
Thus the 3-transistor per bit dynamic memory requires $16 \times 40 \times 16 \times 3 = 30,720$ transistors.

Chip size: $9mm \times 4mm$
Pin count: 18
Transistor count: 50,000

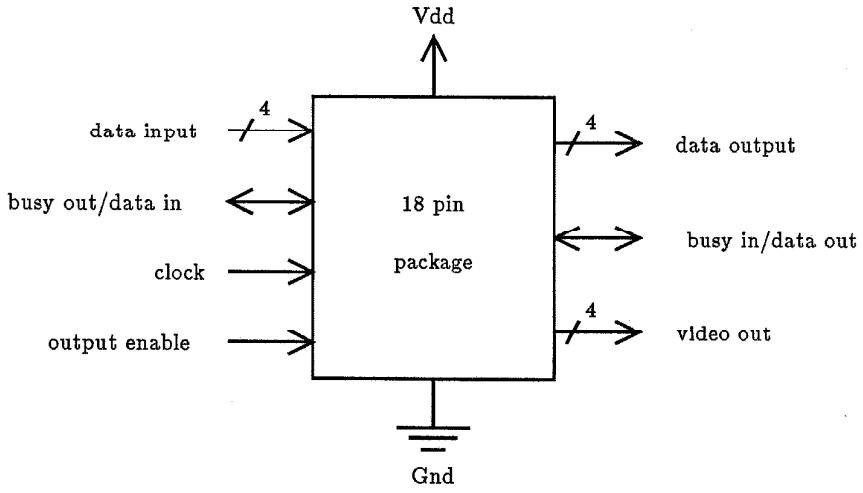


Figure A.1: The PS Chip

A.3 Data Packet Format

All values are integers in the two's complement representation. The format is:

x_{start}	: 12 bits
dx	: 12 bits
y_{scan}	: 12 bits
interpolated values r, g, b	: 8 bits each
interpolated value z	: 16 bits
quotient for r, g, b	: 8 bits each
quotient for z	: 16 bits
remainder for r, g, b	: 8 bits each
remainder for z	: 12 bits
cumulative error for r, g, b	: 12 bits each
cumulative error for z	: 12 bits

This gives us a total of 200 bits of information for each polygon scan line. The interpolation algorithm suggests that the quotient will always have the sign of the value to be interpolated while the remainder will always be positive. The cumulative error has to be compared with the divisor which is always positive. ($\Delta y \geq 0$)

A.4 Current Status

To ease in testing, the chip was logically divided into a number of parts and each was fabricated and tested independently. The division follows closely the chip architecture discussion of Section 3.4.2. The results of testing the chips/subsections are given below:

post-processor	worked
pre-processor	worked
video-processor	not yet tested
memory bank	worked but low yield
processor	works but lacks robustness
clock and timing signal generator	worked at lower speeds ($\approx$ 8Mhz) than designed($\approx$ 40Mhz).

Appendix B

Proof of the Interpolation Algorithm

Objective:
Given the coordinates of the two vertices of an edge, $(x1, y1)$, $(x2, y2)$, we want to find all integer $(x, y) \mid y1 \leq y \leq y2$ that lie on the edge; $x1, y1, x2, y2, x, y$ are integers.

Definition of Terms:

$(x1, y1)$	= one vertex of the edge
$(x2, y2)$	= the second vertex of the edge
$(x_{\text{start}}, y_{\text{start}})$	$= (x1, y1)$ if $y1 \leq y2$
	$= (x2, y2)$ otherwise
$(x_{\text{end}}, y_{\text{end}})$	$= (x2, y2)$ if $y1 \leq y2$
	$= (x1, y1)$ otherwise
Δx	$= x_{\text{end}} - x_{\text{start}}$
Δy	$= y_{\text{end}} - y_{\text{start}} = \mid y1 - y2 \mid$
q	$= \Delta x$ DIV $\Delta y = \lfloor \Delta x / \Delta y \rfloor$
r	$= \Delta x$ MOD $\Delta y = \Delta x - q\Delta y$
x_{real_y}	= the real x-value at $y \mid y1 \leq y < y2 = x_{\text{start}} + (x_{\text{end}} - x_{\text{start}})(y - y_{\text{start}})/\Delta y$
x_y	= the integer x-value at $y \mid y1 \leq y < y2 = \lfloor x_{real_y} \rfloor$
p_{sum_y}	$= \Delta y(x_{real_y} - x_y)$
	$= \Delta y$(the fractional part of x_{real_y}).

Assertion:
$0 \leq p_{\text{sum}_y} < \Delta y$, and

if $p_{\text{sum}_y} + r < \Delta y$, then	x_{y+1}	$= x_y + q;$
	$p_{\text{sum}_{y+1}}$	$= p_{\text{sum}_y} + r$
otherwise	x_{y+1}	$= x_y + q + 1;$
	$p_{\text{sum}_{y+1}}$	$= p_{\text{sum}_y} + r - \Delta y$

PRE condition:

p_{sum}	$= 0;$	% where $p_{\text{sum}}/\Delta y$ is the real cumulative error
count	$= \Delta y \neq 0;$	% the number of scan lines or y-steps
current	$= x_{\text{start}};$	% the x value at $y = y_{\text{start}} + \Delta y - count (= y_{\text{start}})$

POST condition:

p_{sum}	$= 0;$	
count	$= 0;$	
current	$= x_{\text{start}} + \Delta x;$	% the x value at $y = y_{\text{start}} + \Delta y - count (= y_{\text{start}} + \Delta y)$

INVARIANT:

$(\Delta y - count)\Delta x/\Delta y + x_{\text{start}} = current + p_{\text{sum}}/\Delta y;$ % the real x value at $\%y = y_{\text{start}} + (\Delta y - count)$

$0 \leq p_{\text{sum}} < \Delta y;$ % the error is less than 1

INDUCTION STEP:

$$(\Delta y - count + 1)\Delta x/\Delta y + x_{\text{start}} = current + p_{\text{sum}}/\Delta y + \Delta x/\Delta y$$
$$= current + (p_{\text{sum}} + r)/\Delta y + q; \{\Delta x/\Delta y = q + r/\Delta y\}$$

if $(p_{\text{sum}} + r) \geq \Delta y$
then $(p_{\text{sum}} + r)/\Delta y = 1 + (p_{\text{sum}} + r - \Delta y)/\Delta y$
and $0 \leq (p_{\text{sum}} + r - \Delta y) < \Delta y;$

$\rightarrow (\Delta y - count + 1)\Delta x/\Delta y + x_{\text{start}} = current_{\text{new}} + p_{\text{sum}_{\text{new}}}/\Delta y;$
$p_{\text{sum}_{\text{new}}} < \Delta y;$

where	$p_{\text{sum}_{\text{new}}} = p_{\text{sum}} + r;$	$current_{\text{new}} = current + q;$	if $(p_{\text{sum}} + r) < \Delta y$
	$p_{\text{sum}_{\text{new}}} = p_{\text{sum}} + r - \Delta y;$	$current_{\text{new}} = current + q + 1;$	otherwise

Bibliography

[Demetrescu 85] Stefan Demetrescu. High speed image rasterization using scan line access memories. *1985 Chapel Hill Conference on VLSI*, 1985.

[Foley 84] J.D. Foley and A. vanDaam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1984.

[Fuchs 81] Henry Fuchs and John Poulton. Pixel-planes: a vlsi-oriented design for a raster graphics engine. *VLSI Design*, Third Quarter 1981.

[Fuchs 83] Henry Fuchs, John Poulton, Alan Paeth, and Alan Bell. Developing pixel-planes, a smart memory-based raster graphics system. 1983.

[Gharachorloo 85] Nader Gharachorloo and Christopher Pottle. Super buffer: a systolic vlsi graphics engine for real time raster image generation. *1985 Chapel Hill Conference on VLSI*, 1985.

[Gouraud 71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, June 1971.

[Hayes] J.P. Hayes. *Computer Architecture and Organization.*

[Newman 79] William H. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, 1979.

[Obrebska] Monika Obrebska. Comparative survey of different design methodologies for control part of microprocessors. *VLSI Systems and Computations.*

[Poulton et al. 85] John Poulton et al. Pixel-planes: building a vlsi-based graphic system. *1985 Chapel Hill Conference on VLSI*, 1985.